НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.934.2

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Голосовий асистент для інтегрованих середовищ розроблення програмного забезпечення»

Виконав:
студент ІІ курсу, групи КП-81мп
Сахарчук Тарас Юрійович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент
Онай М.В. _____

Консультант з розпізнавання мовлення:
Доцент кафедри ПЗКС, к.т.н., доцент
Сулема Є.С. _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент
Онай М.В. _____

Рецензент:
Старший викладач кафедри ОТ ФІОТ
Виноградов Ю.М. _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (освітня програма) – 121 «Інженерія програмного забезпечення» ("Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем")

ЗАТВЕРДЖУЮ

Науковий керівник кафедри

_____ І.А. Дичка

«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Сахарчуку Тарасу Юрійовичу

1. Тема дисертації «Голосовий асистент для інтегрованих середовищ розроблення програмного забезпечення», науковий керівник дисертації Онай Микола Володимирович, к.т.н., доцент, затверджені наказом по університету від «13» листопада 2019 р. № 3895-С.

2. Термін подання студентом дисертації «16» грудня 2019 р.

3. Об'єкт дослідження: процес використання мовлення для розроблення програмного забезпечення.

4. Предмет дослідження: методи та підходи до вирішення проблем розпізнавання людського мовлення, ідентифікації іменованих сутностей в неструктурованому тексті та генерації програмного коду

5. Перелік завдань, які потрібно розробити:
- обґрунтувати актуальність напрямку досліджень, сформулювати мету та задачі дослідження, ознайомитись з існуючими дослідженнями в обраній галузі;
- здійснити аналіз існуючих рішень;
- сформувати вимоги до голосового асистенту для інтегрованих середовищ розроблення програмного забезпечення;
- здійснити аналіз існуючих підходів до вирішення проблем розпізнавання мовлення, ідентифікації іменованих сутностей, генерації тексту програми;
- оптимізувати існуючі методи та підходи до ідентифікації іменованих сутностей тексту програми;
- розробити масштабовану, модульну архітектуру системи;
- розробити та протестувати програмне забезпечення для голосового асистенту;
- описати бізнес-модель, що дозволить представити на ринку повноцінний програмний продукт із використанням представлених у роботі напрацювань.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:

- загальна архітектура створеної системи;
- схема роботи модифікованого алгоритму розпізнавання іменованих сутностей тексту програми;
- діаграма послідовності «Обробка голосової команди користувача»;
- результати аналізу модифікованого алгоритму розпізнавання іменованих сутностей тексту програми;
- дерево проблем.

7. Орієнтовний перелік публікацій:
   - Тези доповіді “Модифікований метод розпізнавання іменованих сутностей тексту програми ”

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., к.т.н., доцент		
Розпізнавання мовлення	Сулема Є.С., к.т.н., доцент		

9. Дата видачі завдання «25» жовтня 2018 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	24.11.2018	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	11.12.2018	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	08.02.2019	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	12.04.2019	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	22.05.2019	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації	08.06.2019	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу; підготовка матеріалів доповіді на конференції ПМК-2019	07.11.2019	
8.	Оформлення текстової і графічної частини магістерської дисертації	06.12.2019	

Студент Т.Ю. Сахарчук

Науковий керівник дисертації М.В. Онай

РЕФЕРАТ

Актуальність теми. Сфера інформаційних технологій постійно розвивається і безперервно змінюється. На зміну тим технологіям та підходам до розроблення що були актуальні вчора приходять нові засоби та ідеї. Проте майбутнє програмної інженерії це більше ніж просто нові фреймворки, мови програмування або бібліотеки. Це також ті засоби, які власне використовують розробники програмного забезпечення в процесі написання тексту програми. Багато пристроїв, якими ми користуємося кожного дня, містять вбудовані голосові помічники. Більшість існуючих голосових асистентів є рішеннями широкого профілю і не є орієнтованими на вирішення задачі написання тексту програми за допомогою голосу. Саме тому, задача створення голосового асистента для інтегрованих середовищ розроблення програмного забезпечення є актуальною та потребує вирішення.

Об'єктом дослідження є процес використання мовлення для розроблення програмного забезпечення.

Предметом дослідження методи та підходи до вирішення проблем розпізнавання людського мовлення, ідентифікації іменованих сутностей в неструктурованому тексті та генерації тексту програми.

Мета роботи: покращення точності існуючих методів та підходів до ідентифікації іменованих сутностей тексту програми.

Методи дослідження. В даній роботі використовуються методи теоретичного дослідження: аналіз, синтез та узагальнення. Також застосовувалися емпіричні методи: експеримент, спостереження, вимірювання та опис.

Наукова новизна роботи полягає в наступному:

1. Запропоновано модифікований метод розпізнавання іменованих сутностей тексту програми, який полягає в уточненні результатів отриманих стандартними методами шляхом використання

інформації про вже введені сутності в тексті програми та забезпечує підвищення точності на 5% F1 оцінки.

2. Удосконалено існуючі підходи до розпізнавання іменованих сутностей тексту програми шляхом врахування контекстної інформації для обраної предметної області.

Практична цінність. Створений голосовий асистент для інтегрованих середовищ розроблення програмного забезпечення відкриває для розробників наступні можливості:

- Надає можливість написання тексту програми для людей з обмеженими можливостями.
- Підвищує продуктивність розробників, оскільки голос є найбільш ефективним засобом комунікації.
- Дозволяє написання тексту програми за допомогою мобільних пристроїв. Сьогодні, враховуючи особливості наведених платформ, використання інтегрованих середовищ розроблення на них є малопродуктивним або взагалі неможливим, оскільки швидкість введення інформації розробником є досить низькою.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювались на XII науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2019 (Київ, 13-15 листопада 2019 р.).

Структура та обсяг роботи. Магістерська дисертація складається з вступу, п'яти розділів, висновків та додатків.

У вступі надано загальну характеристику роботи, виконано оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, продемонстровано практичну цінність роботи.

У першому розділі проаналізовано існуючі рішення для написання тексту програми за допомогою голосу. Виявлено їх переваги та недоліки. Для проведення аналізу було створено порівняльну таблицю. На основі

досліджень та результатів аналізу було сформовано вимоги до розроблюваної системи.

У другому розділі проведено аналіз існуючих підходів до розпізнавання мовлення, ідентифікації іменованих сутностей, генерації тексту програми. Виявлено їх переваги та недоліки. Для проведення аналізу було створено порівняльну таблицю.

У третьому розділі запропоновано модифікований метод розпізнавання іменованих сутностей тексту програми який враховує особливості обраної предметної області, проведено практичні дослідження та порівняно його швидкодію з існуючими підходами.

У четвертому розділі розроблено архітектуру системи голосового асистенту для інтегрованих середовищ розроблення програмного забезпечення. Наведено детальний опис кожного із компонентів системи. Описано процес тестування системи та шляхи для її подальшого вдосконалення.

У п’ятому розділі сформовано та побудовано бізнес-модель кінцевого продукту.

У висновках проаналізовано отримані результати роботи.

Робота виконана на 77 аркушах, містить 3 додатки та посилання на список використаних літературних джерел з 25 найменувань. У роботі наведено 19 рисунків, 6 таблиць та 2 формули.

Ключові слова: голосовий асистент, мікросервіси, розпізнавання іменованих сутностей.

ABSTRACT

Actuality. The field of information technology is constantly evolving and changing. In place of those technologies and approaches to development that were relevant yesterday, new tools and ideas are coming. However, the future of software engineering is more than just new frameworks, programming languages or libraries. These are also the tools that software developers use in the process of software engineering. Many of the devices we use every day have built-in voice assistants. Most existing voice assistants are broad-based solutions and are not adapted to the process of writing source code by voice. That is why the task of creating a voice assistant for integrated software development environments is urgent and needs to be addressed.

Object of research is the process of software developing by voice.

Subject of research are methods and approaches to solving problems of human speech recognition, identification of named entities in unstructured text and program text generation.

Research objective is to improve the accuracy of existing methods and approaches to identifying named entities of program text.

Research methods. This paper uses methods of theoretical research: analysis, synthesis and generalization. Empirical methods were also used: experiment, observation, measurement and description.

Scientific novelty of the work is as follows:

1. Optimized method for named recognizing of source code entities is proposed, which is to refine the results obtained by standard methods by using information about the entities already entered in the program text and providing an accuracy increase of 5% F1 score.
2. Improved existing approaches to recognizing named entities of program source code by including contextual information for the selected domain area.

The practical value. Created voice assistant for integrated software development environments offers developers the following features:

- Provides the ability to write the program text for people with disabilities.
- Increases developer productivity as voice is the most effective means of communication.
- Allows application text to be written using mobile devices. Today, given the specifics of these platforms, the use of integrated development environments on them is unproductive or impossible at all, since the speed of information input by the developer is quite low.

Approbation. The main provisions and results of the work were presented and discussed at the XII Scientific Conference of Undergraduate and Graduate Students in Applied Mathematics and Computing PMK-2019 (Kyiv, November 13-15, 2019).

Structure and content of the thesis. The master's thesis consists of an introduction, five sections, conclusions and appendices.

The introduction gives a general description of the work, identifies the current state of the problem and the relevance of the research, formulates the purpose and objectives of the research, demonstrates the practical value of the work.

The first section analyzes the existing solutions for writing text using voice. Their advantages and disadvantages are revealed. A comparison table was created for analysis. Based on the research and analysis results, requirements for the developed system were formed.

The second section analyzes the existing approaches to speech recognition, the identification of named entities, and the generation of source code. Their advantages and disadvantages are revealed. A comparison table was created for analysis.

The third section proposes a modified method for recognizing named entities of the program text, which considers the features of the selected domain, compared its performance with existing approaches.

The fourth section describes the architecture of the voice assistant system for integrated software development environments. A detailed description of each of components of the system is given. The process of testing the system and ways for its further improvement are described.

In the fifth section, a business model of the product is created.

The conclusion contains brief overview of the results obtained in the work.

The work is made on 77 sheets, contains 3 appendices and links to the list of used literary sources of 25 titles. The paper includes 19 figures, 7 tables and 2 formulas.

Keywords: voice assistant, named entities recognition, microservices.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

HTTP – протокол передачі гіпертексту;

JSON – формат обміну даними;

STT (Speech to text) – назва групи методів та підходів, які відповідають за конвертацію мови в текст;

NER (Named Entity Recognition) – задача розпізнавання іменованих сутностей в неструктурованому тексті;

SDK (Software Development Kit) – набір засобів для розроблення, які дозволяють спеціалістам створювати застосування до певного пакету програм;

REST – архітектурний стиль, передача стану подання;

IDE – інтегроване середовище розробки програмного забезпечення;

ООП – об’єктно-орієнтоване програмування;

ПЗ – програмне забезпечення;

ПК – персональний комп’ютер;

СУБД – система управління базами даних.

ВСТУП

Останні дослідження в областях машинного навчання та обробки природної мови дали змогу створювати вражаючі речі, які до цього здавались неможливими. Розпізнавання образів та голосових команд, системи рекомендацій, автомобілі з автоматичним керуванням, пошук екзопланет, передбачення шахрайських банківських транзакцій – це лише невеликий список того, що стало можливим завдяки швидкому росту доступної інформації та розвитку вищезгаданих сфер науки.

Одними з таких технологій, які створені для того щоб зробити життя людей простішим є голосові помічники або асистенти – це програмне забезпечення, яке може виконувати завдання або послуги для користувача на основі заданих голосових команд, тобто шляхом обробки та інтерпретації людської мови. Користувачі можуть задавати питання асистентам, контролювати домашні автоматизовані пристрої, відтворювати мультимедіа за допомогою голосу, а також виконувати інші завдання, такі як керування електронною поштою, ведення списку справ, запуск та закриття встановлених на пристрої додатків, тощо за допомогою вербальних команд.

Багато пристроїв, якими ми користуємося кожного дня, містять вбудовані голосові помічники. Вони в наших смартфонах, домашніх аудіо-системах, мобільних програмах та операційних системах. Крім того, технології управління голосом широко застосовуються в автомобілях, електронній комерції, освітніх, медичних та телекомунікаційних середовищах.

Сфера інформаційних технологій постійно розвивається і безперервно змінюється. На зміну тим технологіям та підходам до розроблення що були актуальні вчора приходять нові засоби та ідеї. Проте майбутнє програмної інженерії це більше ніж просто нові фреймворки, мови програмування або бібліотеки. Це також ті засоби, які власне використовують розробники програмного забезпечення в процесі написання тексту програми.

Створення голосового асистенту для інтегрованих середовищ розроблення програмного забезпечення відкриє для розробників наступні можливості:

- Надасть можливість написання тексту програми для людей з обмеженими можливостями. Багато людей, в силу набутих чи вроджених травм, втрачають доступ до розроблення програмного забезпечення і фактично не мають змоги займатись справою яка їм цікава або приносить засоби для існування.
- Підвищить продуктивність розробників, оскільки голос є найбільш ефективним засобом комунікації. Абстрагування від звичайних засобів введення інформації таких як мишка чи клавіатура, дасть змогу програмісту в ефективніший спосіб використовувати власні можливості та можливості комп'ютера. Такий підхід дозволить розробникам виконувати складні дії лише за допомогою однієї розмовної фрази, концентруватись на вирішенні задачі, а не на деталях її реалізації.
- Дозволить написання тексту програми за допомогою мобільних пристроїв. Сьогодні, враховуючи особливості наведених платформ, використання інтегрованих середовищ розроблення на них є малопродуктивним або взагалі неможливим, оскільки швидкість введення інформації розробником є досить низькою.

Отже, задача створення голосового асистента для інтегрованих середовищ розроблення програмного забезпечення є актуальною та потребує вирішення. З огляду на це, сформуємо основні складові науково-технічного дослідження.

Об'єктом дослідження є процес використання мовлення для розроблення програмного забезпечення.

Предметом дослідження є методи та підходи до вирішення проблем розпізнавання людського мовлення, ідентифікації іменованих сутностей в неструктурованому тексті та генерації тексту програми.

Метою наукового дослідження є покращення точності існуючих методів та підходів до розпізнавання мовлення, ідентифікації іменованих сутностей в неструктурованому тексті, генерації тексту програми враховуючи особливості обраної предметної області, а також використання цих методів для побудови голосового асистенту для інтегрованих середовищ розроблення програмного забезпечення.

1. ХАРАКТЕРИСТИКА ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

1.1. Обґрунтування вибору критеріїв оцінювання

Сформуємо критерії для оцінювання існуючих програмних рішень обраної задачі.

Сьогодні існує багато голосових асистентів для вирішення різного кола задач пов’язаних з пошуком інформації, медициною, спрощенням використання пристрою, комерційною діяльністю, тощо. Багато з існуючих варіантів можна в тій чи іншій мірі використовувати для вирішення задачі голосового програмування, незважаючи на те, що якість такого продукту в контексті обраної задачі буде залишати бажати кращого. Через специфіку ринку, лише невелика кількість з наявних сервісів створювалась з метою спрощення роботи з інтегрованими середовищами програмування. Тому одним з перших та основних критеріїв оцінювання існуючих програмних рішень є орієнтованість голосового асистенту на пришвидшення та спрощення процесу написання тексту програми розробником.

Ще одним критерієм оцінювання є те, що система повинна бути з відкритим текстом програми та безкоштовною для використання. Відкрите програмне забезпечення схоже з наукою в тому плані, що дослідження відбуваються в середовищі колегіальної експертної оцінки. Це означає, що наука є в основному відкритим джерелом, що дозволяє ділитись отриманими знаннями з іншими людьми для спільного використання та подальшого розвитку. Програмні рішення, побудовані на принципах доступності та відкритості, дозволяють створювати нові потужніші інструменти, змінювати або комбінувати існуючі залежно від власних потреб. Відкрите програмне забезпечення є важливим, оскільки сприяє розвитку програмної інженерії та є доступним для більш широкого кола користувачів порівняно з платними рішеннями.

Наступною важливою характеристикою продукту є його платформність. Перевагами продуктів-платформ над продуктами

звичайного типу є здатність продукту, бути розширений користувачами на користь інших користувачів шляхом динамічного додавання нової функціональності. Будь-який продукт може стати платформою, надаючи користувачам можливість додавати послуги та функціональні можливості. API є засобами, що сприяють використанню платформи, а також дозволяють користувачам розширювати й додавати послуги над існуючими продуктами. Коли продукт стає платформою, у нього з'являються користувачі нового типу: сторонні розробники.

Іншим важливим критерієм оцінки існуючих рішень є якість документації розробленої системи. Документація програмної системи – це технічний опис продукту, який містить інструкції про те, як ефективно використовувати та інтегруватися з системою. Документація містить всю інформацію, необхідну для роботи з API системи, детальні відомості про функції, типи що вони повертають, аргументи які вони приймають, класи та багато іншого, і зазвичай включають приклади використання тих чи інших функцій. Наявність якісної технічної документації знижує час необхідний для інтеграції з системою, зберігає час необхідний на підтримку існуючої системи, допомагає більшій кількості користувачів познайомитись з продуктом і почати його використання. Для систем які є платформами такі особливості є надзвичайно важливими.

Останнім критерієм оцінки існуючих програмних рішень є здатність до розпізнавання команд сформованих звичайною мовою, а не наперед визначеними фразами. Використання засобів природньої обробки мови дає змогу взаємодіяти з системою за допомогою голосу у звичний для людини спосіб і прибирає необхідність вивчати якийсь новий мовний синтаксис або нові команди для використання програмного рішення. Відповідно ті системи, які використовують такий підхід мають значні переваги порівняно з іншими системи.

В результаті проведеного дослідження були визначені критерії оцінки програмних засобів для створення голосового асистенту, що дозволить

провести детальний порівняльний аналіз існуючих програмних рішень і врахувати їх переваги та недоліки при розробці запропонованих засобів. Розглянуті пункти є тією основою, на якій будуються всі інші аспекти дослідження.

1.2. Аналіз існуючих програмних рішень

1.2.1. *Amazon Alexa*

Alexa є багатофункціональним голосовим асистентом, який розроблюється та підтримується компанією «Amazon». Він включає в себе функції відтворення музики та аудіокниг, встановлення будильників, створення списків завдань, надання інформації про погоду, трафік та інші новини в реальному часі. Користувач може обрати ту чи іншу функцію за допомогою голосової взаємодії з асистентом. Alexa також широко використовується в сфері інтернету речей, де надає можливість керування автоматизованою домашньою системою розумних пристроїв.

Користувачі можуть розширити можливості Alexa шляхом встановлення існуючих або програмування своїх «навичок» – додаткових функціональних можливостей, які створюються зовнішніми розробниками. Користувачі можуть публікувати створені ними «навички» в Alexa Skills систему, а інші користувачі мають змогу використовувати їх в своїх пристроях. Багато навичок запускаються повністю в хмарі з використанням всіх можливостей веб сервісів Амазону.

Amazon також дає змогу розробникам інтегрувати голосові можливості Alexa у свої власні продукти, шляхом надання API інтерфейсу. Продукти, побудовані з використанням цього інтерфейсу, отримують доступ до всіх можливостей голосового асистенту. Alexa забезпечує автоматичне розпізнавання мови за допомогою хмарних сервісів та засобів обробки природної мови. Генерація голосових відповідей асистенту відбувається з використанням засобів глибокого машинного навчання, а саме довгої короткочасної штучної нейронної мережі.

З недоліків цієї системи можна виділити те, що вона є системою широкого призначення і немає готових рішень для вирішення задачі голосового управління системою розроблення програмного забезпечення. Також використання Alexa API не є безкоштовним, відповідно текст програми системи теж не є відкритим.

Перевагами голосового асистенту Alexa є висока точність розпізнавання команд користувачів, модульність системи, можливість додавання власної функціональності для розробників.

1.2.2. *Голосовий помічник «Google»*

Голосовий помічник від компанії «Google» доступний як для мобільних платформ, так і для розумних домашніх пристроїв. Користувачі в основному взаємодіють з Google Assistant за допомогою голосових команд, хоча ввід даних за допомогою клавіатури також є можливим.

Google Assistant надає можливості голосового управління та пошуку, що дозволяє користувачам виконувати широкий спектр завдань: керування розумним домом, доступ до власної особистої інформації з Google сервісів, пошук інформації в інтернеті, бронювання ресторанів поблизу або квитків, перегляд погоди та новин, прослуховування музики, запуск таймерів і нагадувань, організація зустрічей, запуск застосувань но своїх пристроях, надсилання повідомлень, переклад в режимі реального часу та інші.

Google асистент має змогу визначати контекст в якому йому задається питання. Це важливо, оскільки голосовий контроль стає більш потужним і реагує не лише на конкретні фрази або команди, а й враховує попередні питання користувача. У майбутньому, за словами Google, помічник зможе самостійно телефонувати і бронювати зустрічі замість користувачів.

Якщо порівнювати Google Assistant з розглянутим вище голосовим асистентом Amazon Alexa, то обидві ці платформи схожі між собою, оскільки пропонують подібні функції та використовуються на тих же пристроях. Проте, Google має очевидну перевагу, коли мова йде про

операційну систему Android, адже він має доступ до всієї особистої інформації користувачів.

Недоліками цієї системи є те, що вона не є орієнтовано на вирішення задачі голосового управління системою розроблення програмного забезпечення, а надає широкий спектр іншої функціональності. Інтеграція віртуального асистенту Google також не є безкоштовним.

Перевагами голосового Google Assistant є висока точність обробки запитів користувачів, наявність хорошої документації, а також модульність системи.

1.2.3. *Cortrana*

Cortrana є голосовим асистентом створеним компанією Microsoft. Можливості Cortana включають встановлення нагадувань та будильників, розпізнавання голосу користувача, відповідей на запитання (наприклад, надання інформації про поточні результати спортивних змагань, погоду, курси акцій та валют тощо), шляхом використання інформації, отриманої від пошукової системи Bing. Cortrana має можливість запам’ятати голос користувача і таким чином реагувати лише на його команди. Крім голосового введення інформації, голосовий асистент надає можливість клієнтам надавати інформацію за допомогою клавіатури або інших пристроїв введення.

Cortana також надає можливість пошуку файлів та папок і це включає пошук не лише на конкретному пристрої, а й в пов’язаних пристроях на в хмарі, наприклад в сховищі OneDrive. Новини, що надаються голосовим асистентом, будуються з урахуванням інтересів користувача та історії пошукових запитів, таким чином можна помітити що Cortrana інтегрується з рекомендаційною системою від Microsoft. Cortana має змогу перевіряти ваші електронні листи та календар та надсилати голосові сповіщення про них користувачу. Також голосовий асистент має можливість розповідати жарти користувачам.

Cortrana є модульною системою та надає можливість додання власної функціональності для розробників шляхом розроблення та інтеграції «навичок», схожих як у голосовому асистенті від компанії Amazon. Серед інших переваг системи можна виділити підтримка великої кількості мов для взаємодії з голосовим помічником, наявність добре описаної документації та інтеграція з операційною системою Windows.

Недоліками Cortrana є те, що він не орієнтований на вирішення задачі голосового управління інтегрованим середовищем розроблення програмного забезпечення і потребує значних зусиль для реалізації власних «навичок», які будуть надавати згадану функціональність. Іншим недоліком є те, що система не є безкоштовною, тому не всі зацікавлені сторони мають змогу використовувати її.

1.2.4. *VoiceCode.io*

VoiceCode – є голосовим асистентом, який обіцяє не тільки дозволити займатися розробкою програмного забезпечення за допомогою голосу, але й підвищити продуктивність користувачів. Він використовує SmartNav для заміни миші, і під капотом запускає систему Dragon для обробки голосових команд.

Команди, які може обробляти система не є звичними словами розмовної мови, а складаються з спеціальних слів, які зазвичай містять один склад для простішої обробки. Для початку роботи з цією системою користувач повинен ознайомитись з цими командами для подальшого їх використання. Іншим недоліком цієї системи є те, що вона розроблена лише для операційної системи Mac OS, а для найбільш використовуваних платформ таких як Windows або Unix вона не є доступною. Крім того вона не є безкоштовною, оскільки крім плати за власне голосовий асистент потрібно окремо купувати ліцензії на зовнішні системи, що використовуються голосовим асистентом такі як Dragonfly або SmartNav.

Навіть з урахуванням цих недоліків, voicecode.io на даний час є одним із найбільш багатофункціональним рішення для спрощення написання тексту програми голосом.

1.2.5. *Silvius*

Silvius є нащадком Dragon NaturallySpeaking і Aenea. Для розпізнавання мовлення він використовує власний фреймворк, які називається Kaldi. Голосовий асистент може працювати як і в онлайн режимі, так і на невеликих вбудованих пристроях. Silvius працює шляхом передачі вихідного сигналу від мікрофона на сервер, і сервер відповідає змістом тексту, які йому вдалося розпізнати. Отриманий текст потім запускається через граматику, яка і симулюю натискання відповідних клавіш на клавіатурі. Незважаючи на те, що рішення використовує комунікацію з сервером через мережу, голосовий асистент демонструє високу швидкість відгуку.

Сильною інновацією в Silvius є той факт, що він спирається на алгоритм розпізнавання мовлення без прив'язки до конкретної платформи, що дає змогу користуватись голосовим аситентом на будь-якій операційній системі. Ще однією перевагою системи є те, що вона є безкоштовною а її текст програми повністю відкритий для розширення та модифікації. До переваг також можна віднести те, що система є досить простою у встановленні. Під час розроблення була використана мова програмування Python а також фреймворк для аналізу даних – Spark.

В голосовому асистенті Silvius як і VoiceCode команди задаються користувачем не у вигляді звичної розмовної мови, а за допомогою спеціальних слів, що потребує час на їх вивчення. Також, команди що надаються системою не є командами високого рівня і зазвичай зводяться до простого диктування програмного тексту програми розробником. Ще одним недоліком є погана документація, що ускладнює роботу користувачам з цією системою і збільшує час, необхідний для початку роботи з нею.

1.2.6. *Vocola*

Vocola – це мова голосових команд для диктування тексту програми з простим синтаксисом. Багато розробників, які частково паралізовані, навчилися програмувати за допомогою свого голосу завдяки цій системі. Опис доступних команд відбувається за допомогою спеціального словника, в якому створюється відповідність між голосовою командою і клавішами на клавіатурі, які будуть натиснуті.

Vocola є простою у встановлення та використанні, містить простий синтаксис створення команд. Багато існуючих систем розпізнавання голосових команд прив'язуються до конкретної мови програмування. Це означає, що хоча і можна задати будь-яку поведінку системи, але лише на синтаксичному рівні обраної мови програмування. На відміну від цього Vocola розроблена як мова голосових команд, і не є мовою програмування загального призначення. Серед інших переваг системи можна виділити те, що вона є повністю безкоштовною.

Недоліками системи є те, що користувач фактично диктує текст програми голосом, а не дає системі вказівки високого рівня. Як підсумок, система більше підходить для таких завдань як відкриття файлу або вкладки в браузері, ніж для програмування.

1.3. Висновки

Результати проведеного аналізу вказують на те, що жодне з досліджених рішень не задовольняє більшості висунутих вимог. Систематизація оцінок розглянутих програмних засобів представлена в табл. 1.

Як можна помітити із проведеного дослідження, Alexa, Google Assistant і Cortrana є голосовими асистентами широкого профілю і не є орієнтованими на вирішення задачі написання тексту програми за допомогою голосу. Вони є не просто системи, а платформами з якими можуть інтегруватись інші розробники та використовувати їх в своїх

продуктах. Кожна з цих систем має хорошу документацію і підтримуються найбільшими IT компаніями. Проте ще одним недоліком є те, що вони не є безкоштовними і потребують грошових вкладень для їх використання.

Таблиця 1

Порівняння аналогів системи

Назва рішення / Критерій оцінювання	Amazon Alexa	Google Assistant	Cortrana	VoiceCode	Silvius	Vocola
Орієнтованість на написання тексту програми голосом	–	–	–	+	+	+
Не потребує витрат та надає відкритий текст програми	–	–	–	–	+	+
Плаформність або модульність системи	+	+	+	–	–	–
Якість документації	+	+	+	+	–	–
Можливість надання користувачем високорівневих команд	+	+	+	–	–	–

VoiceCode, Silvius та Vocola хоча і є орієнтованими на вирішення поставленої задачі, проте всі вони не надають можливість користувачу вказувати високорівневі команди, і процес їх використання зводиться до простого диктування тексту програми. Також в кожній із систем, команди задаються не розмовною, а власною специфічною мовою, що збільшує час користувачів для початку роботи із системою. Silvius і Vocola є

безкоштовними системами з текстом програми, а VoiceCode потребує витрат для початку роботи із системою.

Отже, проведений аналіз дозволяє зробити висновок про необхідність створення програмних засобів для вирішення задачі управління інтегрованими середовищами розроблення програмного забезпечення, яке не міститиме недоліки розглянутих рішень та використає їх сильні сторони.

2. АНАЛІЗ ПІДХОДІВ ДО ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1. Декомпозиція задачі голосового управління IDE

Для вирішення задачі розроблення голосового асистенту для інтегрованих середовищ розроблення програмного забезпечення потрібно вирішити підзадачі наведені нижче:

- Розпізнавання мовлення. Перетворення мовлення користувача в текст (STT).
- Використання методів обробки природного мовлення користувачів та підходів в області глибокого машинного навчання для виявлення в тексті команд, заданих користувачем, та їх параметрів (ідентифікація сутностей).
- На основі заданої команди та її параметрів генерувати відповідний текст програми для вибраної мови програмування або виконувати відповідну функцію в інтегрованому середовищі розроблення програмного забезпечення.

Детальний аналіз кожної із наведених підзадач та підходів до їх вирішення наведений в наступних розділах.

2.2. Аналіз задачі розпізнавання мовлення

Першою задачею при розробці власного голосового аситенту є задача перетворення мовлення користувача в текст (STT). Розпізнавання мовлення – є процесом конвертації звукового сигналу в цифрові дані, як наприклад текст. Мовлення може подаватись на вхід системи у вигляді файлу або аудіо-потоку.

Сфера розпізнавання мовлення має довгу історію з кількома хвилями значних нововведень. Зовсім недавно ця сфера отримала значний поштовх у розвитку завдяки успіхам в областях глибокого машинного навчання та великих даних. Про розвиток свідчать не лише велика кількість академічних

статей, опублікованих у цій галузі, але що більш важливо використання досягнень у цій сфері в багатьох сферах сучасного життя.

Що стосується доступних програмних ресурсів для вирішення цієї задачі, то такими є розробка «Сфінкс» від університету Карнегі-Меллона, яка надає змогу розпочати вивчати та використовувати техніки розпізнавання мовлення в своїх продуктах. Ще одним таким ресурсом є інструментарій HTK і відповідна йому книга, в якому реалізовано підхід з використанням ланцюгів Маркова до вирішення задачі розпізнавання мовлення. До більш сучасних технологій для вирішення цієї проблеми можна віднести інструментарій Kaldi, який теж є безкоштовно доступним під ліцензією Apache.

Основними методами для вирішення цієї задачі є використання прихованих ланцюгів Маркова або глибоких нейронних мереж, таких як рекурентні нейронні мережі, згорткові нейронні мережі. Детальний аналіз кожного з цих підходів наведений в наступних підпунктах розділу. Для порівняння основних методів та підходів сформує наступні критерії, такі як кількість даних потрібна методу для навчання, швидкість навчання та точність розпізнавання.

2.2.1. *Аналіз прихованої моделі Маркова*

Прихована марковська модель або HMM – це модель, яка базується на статистиці, у якій модельована система розглядається як процес із прихованими станами. У моделях таких як ланцюги Маркова та інших, в яких стан системи є видимим спостерігачеві безпосередньо, і тому єдиними параметрами є ймовірності зміни станів. Щодо прихованої моделі Маркова, то в ній стан не є безпосередньо видимим, але залежно від нього видимим є вихід. Кожен стан містить розподіл ймовірностей усіх допустимих вихідних значень. У підсумку, послідовність значень, що генерується ПММ, дає закодовану інформацію про його послідовність станів. Приховані марковські моделі відомі в першу чергу завдяки їхньому застосуванню в

розпізнаванні часових шаблонів, таких як розпізнавання мовлення, рукописного введення, жестів, морфологічної розмітки, мелодій для акомпонуваня, часткових розрядів та в області біоінформатики.

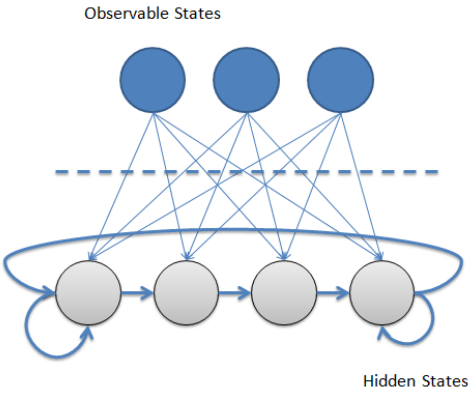


Рис. 1. Діаграма станів у ПММ

Однією з переваг використання цієї моделі для вирішення задачі розпізнавання мовлення є те, що для навчання потрібна невелика кількість даних, оскільки сама задача навчання параметрів у ПММ полягає в знаходженні для заданого набору послідовностей виходів найкращого набору ймовірностей переходів станів та виходів. Такий підхід також показує кращу швидкість навчання порівняно з глибокими нейронними мережами. Серед недоліків такого підходу можна виділити нижчу точність порівняно з іншими методами.

2.2.2. *Аналіз згорткової нейронної мережі*

Згорткові нейронні мережі належать до класу штучних глибоких нейронних мереж з прямим поширенням похибки, які вдало застосовуються до задач розпізнавання мови та образів.

Її особливістю є те, що згорткова нейронна мережа за рахунок застосування згортки мережа зменшує розмір інформації, що міститься в пам'яті, завдяки чому краще справляється із задачами, де присутні велика кількість параметрів.

Процес навчання згорткової нейронної мережі зазвичай інтерпретується як перехід від конкретних особливостей зображення або аудіо до більш абстрактних деталей, і далі до ще більш абстрактним деталей аж до виділення понять високого рівня. При цьому мережа самостійно налаштовується і виробляє сама необхідну ієрархію абстрактних ознак (послідовності карт ознак), фільтруючи незначні деталі і виділяючи лише істотне.

Подібна інтерпретація носить швидше ілюстративний характер. Фактично «ознаки», що виробляються складною мережею, малозрозумілі і важкі для інтерпретації настільки, що в практичних системах не рекомендується намагатися зрозуміти зміст цих ознак, замість цього рекомендується вдосконалити саму структуру і архітектуру мережі, щоб отримати кращі результати.

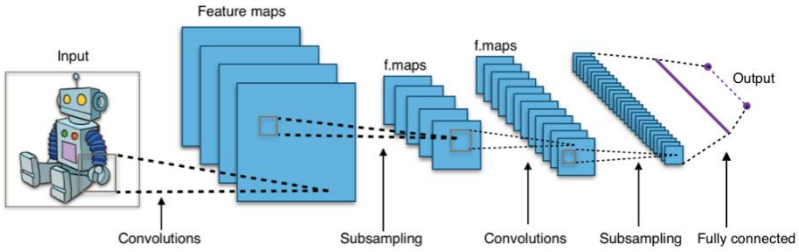


Рис. 2. Архітектура згорткової нейронної мережі

Найбільш популярним і простим способом тренування мережі є навчання з учителем – метод зворотного розповсюдження похибки. Але існує також багато підходів до навчання мережі і без вчителя, тобто без наявності маркованих даних. Для підвищення стійкості мережі, а також покращення її роботи вцілому, запобіганню перенавчання використовується техніка виключення (dropout) – метод тренування підмережі з викиданням випадкових одиничних нейронів.

Перевагами згорткової нейронної мережі є висока точність розпізнавання, навчання за допомогою класичного алгоритму зворотного поширення похибки, зручність використання для паралельних обчислень.

Недоліками глибоких нейронних мереж є довший час навчання порівняно з іншими методами, а також потреба в великій кількості даних для навчання. Ще одним недоліком згорткової нейронної мережі є велика кількість її параметрів, що значно ускладнює конфігурацію потрібну для вирішення конкретної задачі.

2.2.3. *Порівняння підходів до вирішення задачі розпізнавання мови*

У табл. 2, наведено порівняння основних методів та підходів до вирішення задачі перетворення мовлення користувача в текст згідно згаданих вище критеріїв.

Таблиця 2

Порівняння підходів до розпізнавання мовлення

Критерій	Прихована модель Маркова	Згорткова нейрона мережа
Кількість даних потрібних для навчання	Навчається на невеликому наборі даних	Потрібно більше даних для навчання
Швидкість навчання	Швидше навчається порівняно з нейронними мережами	Потрібно значно більше часу для навчання моделі
Точність розпізнавання	Гірші результати порівняно з глибокими нейронними мережами	Демонструє кращі результати в загальному випадку

У підсумку можна помітити, що глибокі нейронні мережі дають кращу точність розпізнавання, але для їх навчання потрібно більше ресурсів, а саме часу та даних.

2.3. Аналіз задачі ідентифікації іменованих сутностей в тексті

Розпізнавання іменованих сутностей – це підзадача отримання інформації, яка прагне знайти описані сутності в неструктурованому тексті і класифікувати їх до однієї з наперед визначених категорій, такі як імена осіб, місця, медичні коди, часові вирази, гроші, відсотки тощо.

До основних програмних рішень цієї задачі належать:

- GATE підтримує NER на багатьох мовах і доменах та надає можливість використання графічного інтерфейсу і Java API.
- OpenNLP включає в себе як засноване на правилах, так і статистичне розпізнавання іменованих сутностей.
- SpaCy має швидкий статистичний NER, а також візуалізатор іменованих сутностей та є системою з відкритим текстом програми.

Системи розпізнавання іменованих сутностей в тексті зазвичай створюються з використанням лінгвістичних методів граматики, а також статистичних моделей та машинного навчання. Аналіз підходів та методів вирішення задачі ідентифікації іменованих сутностей в тексті наведений в наступних пунктах. Порівняння методів відбувається за такими критеріями як кількість даних потрібних для навчання, точність класифікації, час навчання та здатність навчання моделі на нових даних.

2.3.1. *Аналіз рекурентної нейронної мережі*

Рекурентні мережі належать до класу штучних глибоких нейронних мереж, у яких зв'язки між вузлами є орієнтованим графом. Це дає змогу створити стан всередині мережі, а також використовувати її внутрішню пам'ять для аналізу будь-яких послідовностей залежностей між їх станами та входів. Рекурентна нейронна мережа має можливість зберігати інформацію, отриману на попередніх етапах навчання, і ця інформація впливає на рішення моделі в конкретний момент часу.

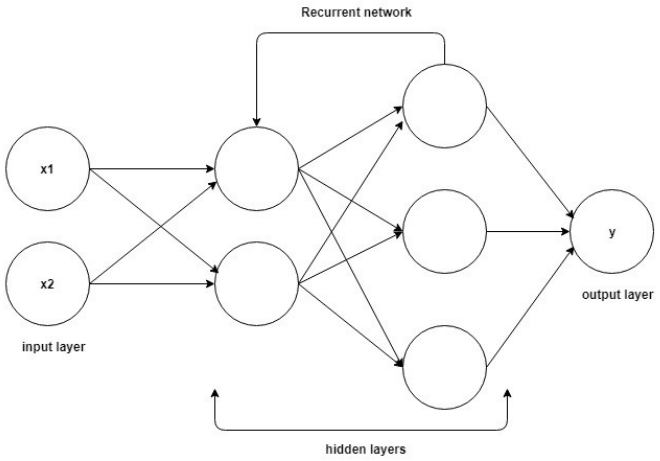


Рис. 3. Архітектура рекурентної нейронної мережі

На відміну від нейронних мереж з прямим поширенням похибки, рекурентні мережі мають змогу застосовувати свою внутрішню пам'ять для аналізу будь-яких вхідних послідовностей. Це дозволяє їх використовувати до вирішення таких задач, як розпізнавання неперервного тексту та мовлення вцілому. Основними бібліотеками які надають готову реалізацію глибоких нейронних мереж є Caffe, Keras, Theano, TensorFlow, Torch.

Перевагами такого підходу є висока точність ідентифікації іменованих сутностей в неструктурованому тексті. Серед недоліків можна виділити потребу в великій кількості даних для навчання і час, необхідний для тренування мережі на цих даних.

2.3.2. *Аналіз умовних випадкових полів (CRF)*

Умовне випадкове поле – це статистичний метод класифікації, характерною особливістю якого є можливість враховувати контекст об’єкта, що класифікується. Алгоритм відноситься до методів машинного навчання з учителем. Даний метод знайшов широке застосування в різних областях, зокрема, його успішно використовують в завданнях розпізнавання мови і образів, обробки текстової інформації, біоінформатиці, комп'ютерній графіці та ін.

Умовне випадкове поле є типом неорієнтованої ймовірнісної графічної моделі. Він використовуються для кодування відомих

взаємозв'язків між спостереженнями і часто використовуються для маркування або розбору послідовних даних, таких як обробка природної мови, біологічних послідовностей, в сфері комп'ютерного зору. Також CRF є альтернативою відповідним прихованим марковським моделям (HMM) . У комп'ютерному зорі CRF часто використовуються для розпізнавання об'єктів і сегментації зображень.

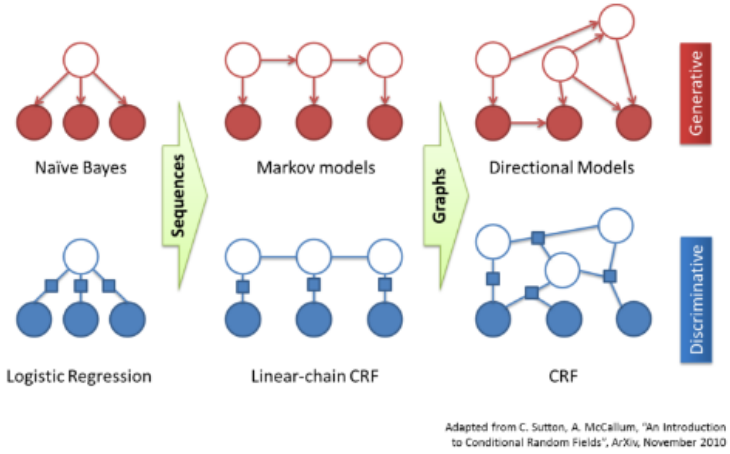


Рис. 4. Типи випадкових умовних полів

Перевагою використання методу умовних випадкових полів є висока точність. Найбільш вагомим недоліком такого підходу є те, що складно тренувати існуючу модель на нових даних та статистичні системи зазвичай вимагають великого обсягу вручну анотованих навчальних даних. Крім того модель не може працювати з незнайомими словами, які не були присутні в датасеті для початкового навчання.

2.3.3. *Аналіз підходу з ручним пошуком шаблонів в реченнях*

Цей підхід до пошуку іменованих сутностей в неструктурованому тексті полягає в ручному аналізі доменної області та визначенні шаблонів на основі яких буде відбуватись ідентифікація іменованих сутностей.

Системи граматики, створені вручну, зазвичай отримують більшу точність, але ціною меншого запам'ятовування і місяцями роботи досвідчених лінгвістів. Для успішного використання такого підходу

необхідна наявність статистики вживання мовних конструкцій в реченнях. Ручний аналіз текстів використовується при обробці текстів обмеженої тематики.

В загальному випадку розроблену модель складно використати в іншій доменній області, а також потрібен значний час на перенавчання моделі. Перевагою даного підходу є висока точність ідентифікації іменованих сутностей.

2.3.4. *Порівняння методів ідентифікації іменованих сутностей*

Порівняння переваг та недоліків розглянути методів та підходів до ідентифікації іменованих сутностей в неструктурованому тексті наведені в табл. 3.

Згідно аналізу рекурентні нейронні мережі розпочали показувати кращі результати порівняно з іншими методами в останні роки через наявність великої кількість даних. Зазвичай використання умовних випадкових полів було основним вибором для вирішення задачі ідентифікації іменованих сутностей. Підхід з ручних аналізом тексту теж показує високі результати, але вимагає знання в доменній області та лінгвістиці. Крім того, всі розглянуті методи крім нейронних мереж не мають здатності до навчання моделі на нових даних.

Таблиця 3

Порівняння методів ідентифікації іменованих сутностей

Критерії	Рекурентна нейронна мережа	Ручний пошук шаблонів в тексті	Умовні випадкові поля
Кількість даних потрібних для навчання	Потрібний великий набір даних для навчання	Потрібні знання в лінгвістиці	Потрібна менша кількість даних
Час навчання	Потрібен значний час для навчання	Потрібен значний час для аналізу доменної області	Дещо швидший час навчання порівняно з рекурентними мережами
Точність	В середньому точність краща на 1-2% порівняно з іншими методами	Залежить від доменної області, зазвичай демонструє високу точність	Висока точність ідентифікації
Здатність навчання моделі на нових даних	Так	Ні	Ні

2.4. Аналіз процесу виконання команд користувача

В розробленій системі команди користувача будуть двох видів:

- Команди, призначені для керування інтегрованим середовищем розроблення програмного забезпечення.
- Команди, призначені на власне написання тексту програми розробником.

Для виконання команд першого типу необхідно реалізувати інтеграцію системи з обраним середовищем розроблення програмного забезпечення. Майже всі сучасні IDE мають доступний та добре документований API для можливості використання тих чи інших функцій.

Процес виклику користувачем будь-якої функції інтегрованого середовища за допомогою голосового асистенту виглядає наступним чином:

3. Голосовий асистент отримує команду від користувача.
4. Система розпізнає команду та параметри до неї шляхом ідентифікації іменованих сутностей.
5. Система шляхом виклику відповідного методу інтерфейсу інтегрованого середовища розроблення програмного забезпечення виконує задану команду користувача.

Для виконання команд другого типу, які потребують генерування тексту програми буде використано абстрактне синтаксичне дерево.

Абстрактне синтаксичне дерево (AST) є представленням абстрактної синтаксичної структури вихідного тексту програми. Кожен вузол дерева позначає елемент, що міститься у вихідному тексті програми. Наприклад, групові дужки не зображуються явно в структурі дерева, а синтаксична конструкція if-then може бути позначена за допомогою одного вузла з трьома гілками. Це відрізняє абстрактні синтаксичні дерева від конкретних синтаксичних дерев, які зазвичай будуються парсером під час процесу перекладу та компіляції вихідного тексту програми. Абстрактне синтаксичне дерево крім того дає змогу зберігати контекстну інформацію та виконувати контекстний аналіз. Анотація синтаксичних дерев також використовується в системах аналізу і перетворення програм.

2.5. Висновки

В даному розділі здійснено аналіз існуючих підходів до розпізнавання мовлення, ідентифікації іменованих сутностей, генерації тексту програми, а також виявлено їх переваги та недоліки. Для проведення аналізу було створено порівняльні таблиці. На основі проведеного аналізу знайдено шляхи для удосконалення існуючих підходів шляхом врахування особливостей обраної доменної області.

3. МОДИФІКОВАНИЙ МЕТОД РОЗПІЗНАВАННЯ ІМЕНОВАНИХ СУТНОСТЕЙ

Науковою складовою даного дослідження є оптимізація існуючих методів та підходів до розпізнавання мовлення, ідентифікації іменованих сутностей в неструктурованому тексті, генерації тексту програми враховуючи особливості обраної предметної області. В цьому розділі запропоновано модифікацію існуючих методів розпізнавання іменованих сутностей в неструктурованому тексті в контексті задачі побудови голосового асистенту для інтегрованих середовищ розроблення програмного забезпечення.

Задача розпізнавання іменованих сутностей тексту програми виникає при створенні голосового асистенту для спрощення розроблення програмного забезпечення. Ідентифікацію та категоризацію іменованих сутностей необхідно здійснювати для розуміння системою змісту команд заданих користувачем системи.

Задача виявлення іменованих сутностей є розділом обробки природної мови та полягає у класифікації та пошуку в неструктурованому тексті словосполучень або слів, що складають інтерес в контексті поставленої задачі. Причому знайдені частини тексту розподіляються у заздалегідь визначені типи категорій, а в ролі іменованих сутностей можуть виступати імена конкретних людей, назви компаній або локацій, валюти тощо. Підходи та методи для вирішення поставленої проблеми широко застосовуються при побудові голосових асистентів, отриманні інформації з тексту, машинному перекладі.

Хоча існуючі методи виявлення іменованих сутностей показують досить непогані результати їх можна значною мірою покращити шляхом врахування особливостей доменної області де вони застосовуються. В даному розділі розглянуто використання NER в контексті задачі розроблення голосового асистенту для пришвидшення та спрощення

написання текстів програм. В даній доменній області виділені наступні іменовані сутності, а саме: назви змінних, класів, функцій, файлів текстів програм.

Використання запропонованої оптимізації існуючих методів та підходів до виявлення іменованих сутностей (наведені в підрозділі 2.3) дозволить підвищити точність розпізнавання команд користувача, відповідно і точність голосового асистента вцілому. Це дасть змогу побудувати голосовий асистент, що спростить процес написання тексту програм та підвищить продуктивність розроблення програмного забезпечення.

3.1. Опис запропонованої модифікації існуючих методів

Всі команди що надходять від користувача голосовому асистенту умовно можна поділити на два типи.

До першого типу належать такі команди де користувач вносить нову іменовану сутність у вже існуючий контекст. Наприклад, команда «Створи нову змінну типом string та назвою name» містить наступні іменовані сутності, а саме тип змінної («string») та її назву («name»).

До другого типу належать команди в яких користувач звертається до вже існуючих іменованих сутностей в контексті тексту програми – створених або існуючих в стандартній бібліотеці назв змінних, функцій, модулів, операторів, файлів тощо. Команда «Виконай функцію time» містить одну іменовану сутність, а саме назву функції «time». Очікується, що функція з такою назвою вже існує в контексті програми, оскільки користувач намагається її викликати. В процесі написання тексту програми розробником, очевидно, що він буде набагато частіше звертатися до існуючих функцій, змінних, класів ніж створювати нові. Саме тому команди другого типу будуть вживатись користувачами голосового асистенту набагато частіше порівняно з командами першого типу. Відповідно покращення точності розпізнавання іменованих сутностей в командах цього

типу значно вплинуть на точність роботи голосового асистента загалом. Тому в даній роботі приділимо основну уваги покращенню точності розпізнавання команд другого типу.

Для покращення точності розпізнавання іменованих сутностей в командах другого типу будемо використовувати інформацію, яка наявна в контексті тексту програми, а саме доступні назви змінних, операторів, функцій, модулів, файлів тощо. Модифікований метод виглядає наступним чином.

1. Етап попередньої обробки. На цьому етапі виконується аналіз тексту програми та виявлення існуючих іменованих сутностей. Для цього доцільно використовувати синтаксичний аналізатор, який на виході побудує абстрактне синтаксичне дерево. Проаналізувавши дерево можна виокремити в окремі структури даних назви змінних, функцій, модулів які існують в тексті програми.
2. На другому етапі необхідно застосувати один із стандартних методів, які були наведені вище, для ідентифікації іменованих сутностей в команді заданій користувачем. На виході методу ми отримаємо тип команди, яку вказав користувач, і потенційні іменовані сутності які розпізнав використаний метод.
3. На третьому етапі виконується перевірка типу команди, яку розпізнав стандартний метод. Якщо команда належить до другого типу, тоді переходимо на наступний етап, в іншому випадку повертаємо іменовані сутності що були розпізнані стандартним методом та завершуємо роботу алгоритму.
4. Уточнюємо виявлені іменовані сутності шляхом знаходження відповідної існуючою сутності в списках, створених на першому етапі. Пошук доцільно виконувати не методом повного співпадіння, а за допомогою введення функції схожості (similarity function) між двома сутностями. Наприклад, для побудови такої функції можна використовувати Soundex алгоритм, що дає змогу

знаходити фонетично близькі слова. В такому випадку для розпізнаної стандартним методом іменованої сутності буде підібрана найбільш схожа по звучанню існуюча сутність в тексті програми.

5. Завершуємо алгоритм та повертаємо знайдену існуючу іменовану сутність в тексті програми, яка має найвище значення функції схожості (similarity function) із сутністю-кандидатом, що була розпізнана стандартним методом.

Запропонований модифікований метод зображений на рис. 5.

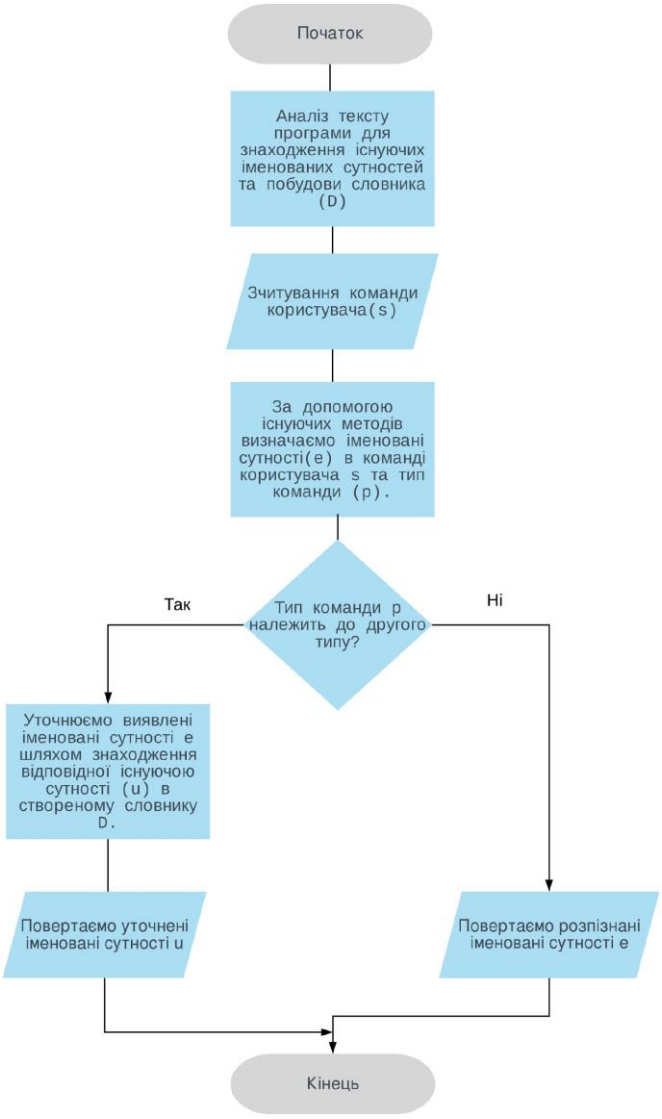


Рис. 5. Алгоритм реалізації модифікованого методу розпізнавання іменованих сутностей

3.2. Аналіз функцій схожості для порівняння іменованих сутностей

Як вже було зазначено в підрозділі 3.1 на четвертому етапі модифікованого методу розпізнавання іменованих сутностей для знаходження відповідної іменованої сутності в словнику доцільно використовувати не метод повного співпадіння двох сутностей, а шляхом введення функції схожості між двома іменованими сутностями. В цьому підрозділі розглянуто існуючі метрики для порівняння сутностей та їх застосування в контексті даної задачі.

3.2.1. *Відстань Левенштейна*

Відстань Левенштейна – це метрика, яка являє собою мінімальну кількість змін символів, потрібну щоб перетворити одне слово в інше. Відповідно результатом цієї метрики є додатне число, яке залежить від довжини вхідних рядків. Наприклад, відстань Левенштейна між словами “море” та “гори” становитиме 3, оскільки задані слова відрізняються трьома символами. Метрика широко застосовується для перевірки орфографії, систем корекції оптичного розпізнавання символів та в процесі перекладу тексту з однієї мови на іншу. Недоліком методу є те, що його результат залежить від довжини слів що порівнюються, тому доцільно виконувати нормалізацію результату відносно довжини більшого слова.

3.2.2. *Індекс Джакарта*

Також для порівняння двох слів можна використовувати індекс Джакарта, як є однією із найперших мір подібності порівнюваних об’єктів і активно використовується в інформатиці, біології, хімії, екології і інших напрямках. В загальному випадку індекс Джакарта задається формулою (1).

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

У даній формулі в ролі дільника виступає кількість спільних токенів (перетин) слів що порівнюються, а в ролі діленого загальна кількість

унікальних токенів (об’єднання). Як видно з формули, обрахований коефіцієнт може набирати значення від 0 до 1. Чим вищий показник подібності Джакарта отриманий за формулою (1), тим більше слова що порівнюються подібні між собою. В ролі токенів можуть виступати послідовності символів довільної довжини.

3.2.3. *Метод Ратклiффа-Обершельпа*

Ще одним підходом до порівняння схожості іменованих сутностей є метод Ратклiффа-Обершельпа, який належить до групи алгоритмів на основі послідовностей символів. Ідея алгоритму полягає у знаходження найдовшої спільної підпослідовності слів що порівнюються. Після цього знайдена частина видаляється з обох рядків і відповідно кожен з них розбивається на дві частини відносно місця видалення. На наступному кроці цей же метод використовується для лівої частини, отриманої на попередньому етапі, а також для правої. Цей процес повторюється рекурсивно доти, поки розмір отриманої підпослідовності стане меншим за критичне значення, яке можна обирати або задавати як параметр методу. Оцінка, що повертається методом рівна подвоєному добутку довжини спільної підпослідовності поділена на суму довжин рядків, що порівнюються. Відповідно результат методу Ратклiффа-Обершельпа лежить в межах від 0 до 1 і чим більше це значення, тим більш схожі рядки що порівнюються.

3.2.4. *Soundex алгоритм*

Також для порівняння схожості іменованих сутностей використовується Soundex алгоритм, який використовує фонетику для оцінки схожості. Тобто слова які звучать майже однаково, хоча й пишуться по різному будуть мати досить високе значення цієї метрики. Такий підхід досить корисний при побудові голосового асистенту, адже дозволяє усунути помилки які були отримані на попередніх етапах роботи системи, а саме при розпізнаванні голосової команди користувача та конвертації її в текст.

Soundex алгоритм є найбільш популярних із групи фонетичних алгоритмів і реалізований в таких системах керування базами даних як Oracle, DB2, MySQL, PostgreSQL та інших.

Алгоритм полягає у кодуванні вхідного слова, при чому переважно кодуються приголосні звуки, а голосні не враховуються за виключенням першого. При порівнянні слів – порівнюються отримані для них коди. Код складається з літери і трьох числових розрядів – першої букви слова і закодованих наступних приголосних.

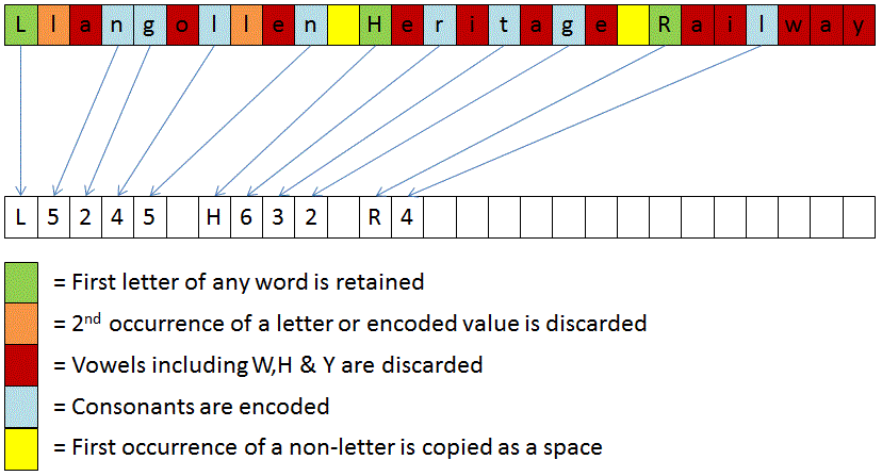


Рис. 6. Приклад кодування слів Soundex алгоритмом

Подібні приголосні кодуються однією й тією ж цифрою. Голосні, хоча і можуть вплинути на результат, проте не кодуються крім першої літери. На рис. 6 зображено приклад кодування речення використовуючи Soundex алгоритм.

Враховуючи особливості розглянутих алгоритмів в контексті застосування його для уточнення іменованих сутностей при побудові голосового асистента доцільно використовувати Soundex алгоритм, оскільки порівняно з іншими розглянутими підходами, він дає змогу усунути неточності, які були отримані на попередніх етапах роботи системи, а саме при конвертації голосової команди користувача в текст.

3.3. Практичні дослідження

Для знаходження іменованих сутностей базовим методом було використано мову програмування Python, а також бібліотеки spaCy та NLTK.

NLTK (Natural Language Toolkit) – це пакет, написаний на мові програмування Python, який надає інструментарій для обробки текстів та використання алгоритмів природної обробки мови. Для вирішення задачі розпізнавання іменованих сутностей цей пакет використовує ефективну реалізацію умовних випадкових полів (CRF), який відноситься до статистичних методів. NLTK супроводжується детальною документацією і покрита тестами.

SpaCy популярна бібліотека для обробки природньої мови та аналізу текстів для Python. Вона написана за допомогою Cython, яка є видом мови програмування Python проте з швидкодією як в C. Саме через це багато компаній використовують її при розробці власних проектів, в той час як NLTK використовують в навчальних цілях. Бібліотека містить засоби для токенізації та класифікації текстів, визначення частин мови для слів, визначення залежності між словами в реченні, розпізнавання іменованих сутностей, а інтеграцію із засобами глибокого машинного навчання і візуалізацію даних.

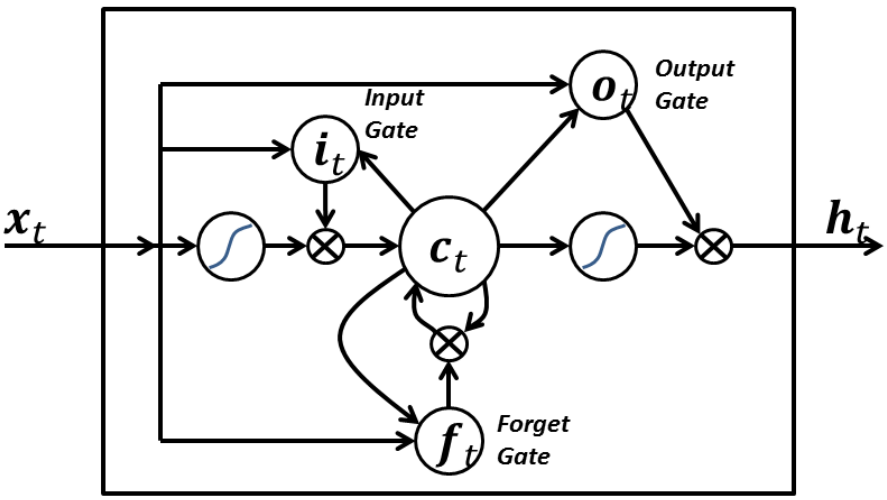


Рис. 7. Архітектура LSTM блоку

SpaCy дозволяє вирішити поставлену задачу як за допомогою статистичних методів так і методами глибокого машинного навчання, а саме

з використанням глибокої нейронної мережі LSTM (Long Short-Term Memory), зображеної на рис. 7.

Для виявлення існуючих іменованих сутностей в тексті програми для подальшого їх уточнення використовувався модуль ast.AST (Abstract Syntax Tree). Цей модуль дозволяє представляти програмний текст програми написаний на вхідній мові програмування у вигляді синтаксичного дерева, що спрощує подальший його аналіз. Кожен вузол цього дерева відповідає операторам мови програмування, а листки цього дерева є операндами. Абстрактне синтаксичне дерево не відображає весь текст програми повністю, але лише ту інформацію, якої достатньо для аналізу тексту програми. Приклад абстрактного синтаксичного дерева, побудованого на основі вхідного тексту програми, зображений на рис. 8.

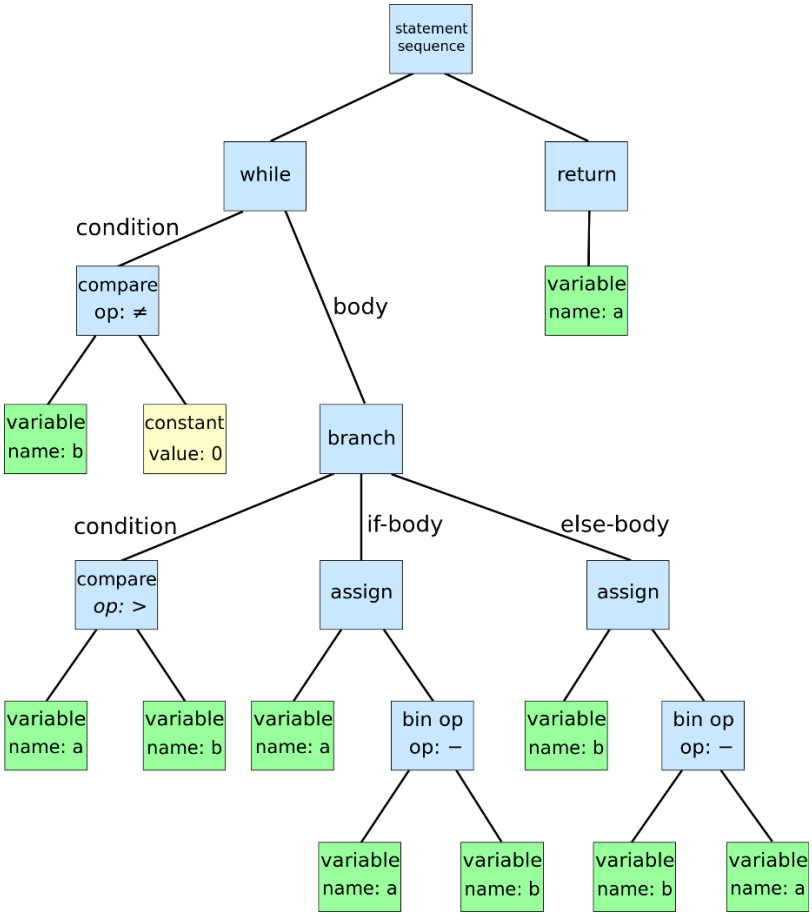


Рис. 8. Абстрактне синтаксичне дерево

Модуль ast.AST є вбудованим в стандартну бібліотеку мови програмування Python та спрощує роботу з абстрактними синтаксичними

деревами. Головним призначенням цього модуля є представлення граматики вхідного тексту програми у вигляді синтаксичного дерева.

Для оцінки точності алгоритмів було використано метрику F1. При оцінці вона враховує точність (precision) та повноту (recall) прогнозів, зроблених методами. Точність (p) – це відношення кількості елементів, які були правильно розпізнані як іменовані сутності, до всієї кількості елементів які алгоритм розпізнав як іменовані сутності. Повнота (r) – це відношення кількості елементів, які були правильно розпізнані як іменовані сутності, до всієї кількості іменованих сутностей які повинні були бути розпізнані алгоритмом.

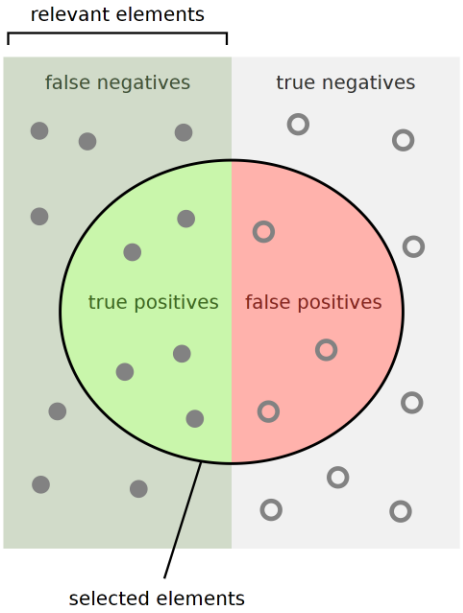


Рис. 9. Візуалізація F1 оцінки

F1 оцінка є середнім гармонічним між точністю розпізнавання та повнотою. Ця метрика досягає найкращого значення при 1 і найгіршого при 0. Для обрахунку даної метрики потрібно використовувати формулу (2), де p – це точність; r – повнота.

$$F_1 = 2 \cdot \frac{p \cdot r}{p + r} \tag{2}$$

Для дослідження було використано 3 набори даних з наступними характеристиками:

- малого розміру (Small) – містить 30 тестових команд користувача та 3 файли тексту програми;
- середнього розміру (Medium) – містить 100 тестових команд користувача та 22 файлів тексту програми;
- великого розміру (Large) – містить 450 тестових команд користувача та 61 файл тексту програми.

Кожен з наборів був розділений на вибірку для навчання і вибірку для тестування у відношення 75% та 25% для забезпечення об’єктивної оцінки роботи алгоритмів. Також набори даних містили по 5 типів токенів (variable_name, function_name, module_name, class_name, command_type).

На рис. 10 наведені отримані практичні результати точності алгоритму умовних випадкових полів з використанням бібліотеки NLTK та запропонованої оптимізації.

Для статистичного алгоритму були отримані наступні результати оцінки F1: 84.17 – малий набір даний, 85.64 – середній, 87.23 – великий. Результати статистичного алгоритму з використанням запропонованої оптимізації наступні: 87.46 – малий набір даний, 93.75 – середній, 89.09 – великий.

Для алгоритму глибокого машинного навчання, а конкретно для архітектури нейронної мережі з використанням LSTM та бібліотеки SpaCy, були отримані наступні результати оцінки F1: 81.12 – малий набір даний, 83.43 – середній, 89.77 – великий. Після використання запропонованої модифікації та виконання етапу уточнення розпізнаних іменованих сутностей було отримано наступні результати: 85.61 – малий набір даний, 93.41 – середній, 91.22 – великий. Порівняння точності обраного алгоритму глибокого машинного навчання із запропонованою оптимізацією наведено на рис. 11.

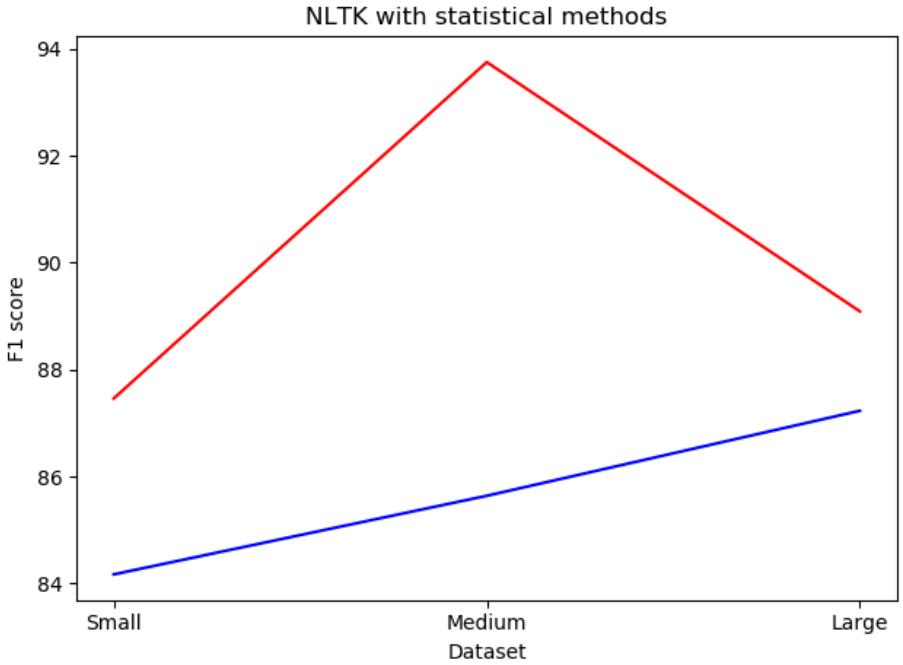


Рис. 10. Порівняння роботи алгоритму випадкових умовних полів (синій) та запропонованої оптимізації (червоний)

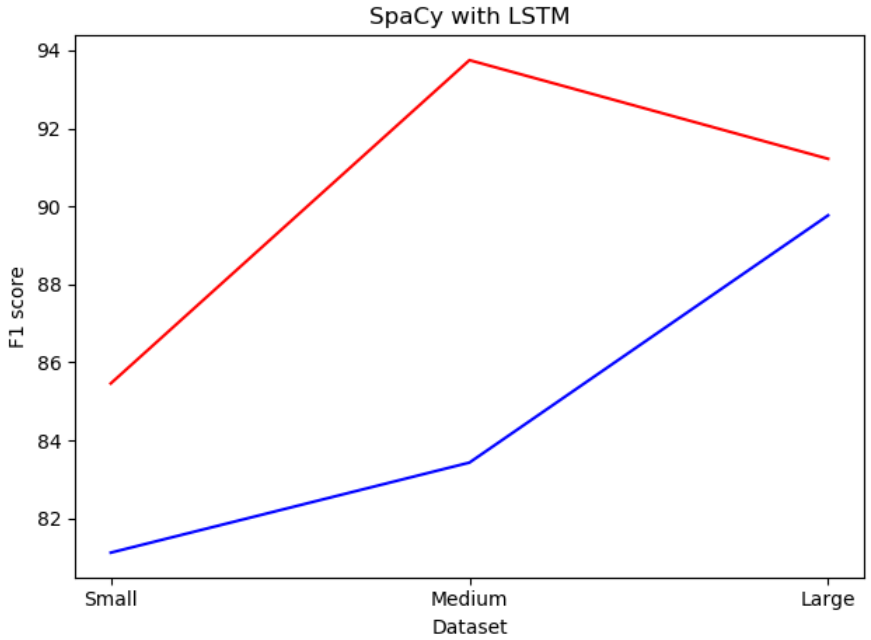


Рис. 11. Порівняння роботи методів глибокого машинного навчання (синій) та запропонованої оптимізації (червоний)

Згідно отриманих практичних результатів можна помітити, що методи після використання запропонованої оптимізації, а саме етапу уточнення

розпізнаних іменованих сутностей демонструють кращу оцінку точності на кожному з наборів даних. Покращення точності розпізнавання для модифікованого методу порівняно з базовим варіантом для статистичного алгоритму становить від 1.86 до 8.11 оцінки F1. Якщо в якості базового алгоритму обрати методи глибокого машинного навчання, то покращення точності лежить в межах від 1.45 до 10.31 метрики F1.

Порівнявши точність базових алгоритмів, а саме методу випадкових умовних полів (CRF), який належить до групи статистичних методі, та нейронної мережі з використанням LSTM, яка належить до методів глибокого машинного навчання, можна помітити що перший дає дещо кращі результати на малому та середньому наборах даних, а другий демонструє кращу точність на великому наборі даних. Це є закономірним, оскільки методи глибокого машинного навчання потребують більше даних для навчання порівняно з статистичними методами і на більших наборах даних їх точність є вищою.

Проаналізувавши графіки для модифікованих методів на рис. 9 та рис. 10, можна що приріст точності отриманої порівняно з базовим алгоритмом, є вищим для середнього набору даних ніж для малого або великого. Для малого набору даних приріст точності є нищим порівняно із середнім, оскільки текст програми є невеликим і відповідно містить невелику кількість існуючих іменованих сутностей, які можна використовувати для уточнення базового алгоритму. Для великого набору даних ситуація протилежна. Оскільки, в контексті тексту всієї програми міститься велика кількість існуючих іменованих сутностей які схожі між собою, тому на етапі уточнення також може обиратись невірна іменована сутність. Тому при побудові голосового асистента доцільно не будувати словник іменованих сутностей для всього тексту програми, а використовувати динамічний контекст на основі тих іменованих сутностей які існують в області видимості користувача або в стандартній бібліотеці, тобто враховувати лише ті іменовані сутності які доступні для використання

користувачем. Отримання лише тих сутностей, які доступні в області видимості користувача, можна здійснювати з використанням SDK інтегрованого середовища розроблення для якого будується голосовий асистент. Саме такий підхід і використаний при розробленні голосового асистенту в даній магістерській дипломній роботі.

3.4. Висновки

В даному розділі запропоновано модифікований алгоритм для вирішення задачі ідентифікації іменованих сутностей тексту програми, який враховує особливості обраної доменної області та дозволяє підвищити точність розпізнавання іменованих сутностей на 5% в середньому порівняно із статистичними підходами та підходами глибокого машинного навчання. Також здійснено аналіз функцій схожості іменованих сутностей для використання на етапі їх уточнення.

4. АНАЛІЗ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Огляд архітектури програмного забезпечення

В якості архітектурного підходу до розроблення голосового асистенту для інтегрованих середовищ було обрано мікросервісний підхід. З використанням мікросервісів система розбивається на невеликі функціональні компоненти, які незалежні один від одного. Порівняно з традиційною монолітною архітектурою, в якій система складалась з єдиного великого модулю, мікросервіси розділені і працюють разом щоб виконувати те ж завдання.

Завдяки використанню такого підходу система отримає наступні переваги:

- Швидкий час розгортання та внесення змін до конкретного сервісу. Також ці процеси стають більш керованими та передбачуваними порівняно з традиційними підходами.
- Важливою перевагою з точки зору розробки продукту, який використовує моделі машинного навчання є те, що кожен мікросервіс може бути написаний з використанням технологій, які найкраще підходять для вирішення задачі за яку він відповідає. При створенні голосового асистенту для вирішення задачі розпізнавання та обробки тексту доцільно використовувати бібліотеки та мову програмування Python, а ось для логування інформації краще підходить ELK стек який написаний на мові програмування Java.
- Кожен із сервісів можна масштабувати горизонтально, що підвищує швидкодію та стійкість системи, адже вона зможе коректно працювати навіть у випадку якщо конкретний екземпляр сервісу вийде з ладу.

- Оскільки сервіси не є тісно зв’язаними між собою і взаємодіють через API інтерфейс, що дозволяє замінювати реалізації сервісів незалежно від інших.

Архітектура спроектованої системи голосового асистенту для інтегрованих середовищ розроблення зображена на рис. 12 та складається з 7 сервісів: клієнт – плагін для конкретного середовища розроблення програмного забезпечення (Client), сервіс розпізнавання мовлення користувача та конвертації його в текст (Speech Recognition service), сервіс ідентифікації іменованих сутностей (Named Entities Recognition service), сервіс логування (Logging service), сервіс конфігурації (Configuration service), сервіс виявлення сервісів (Discovery service) та API шлюзу (API Gateway).

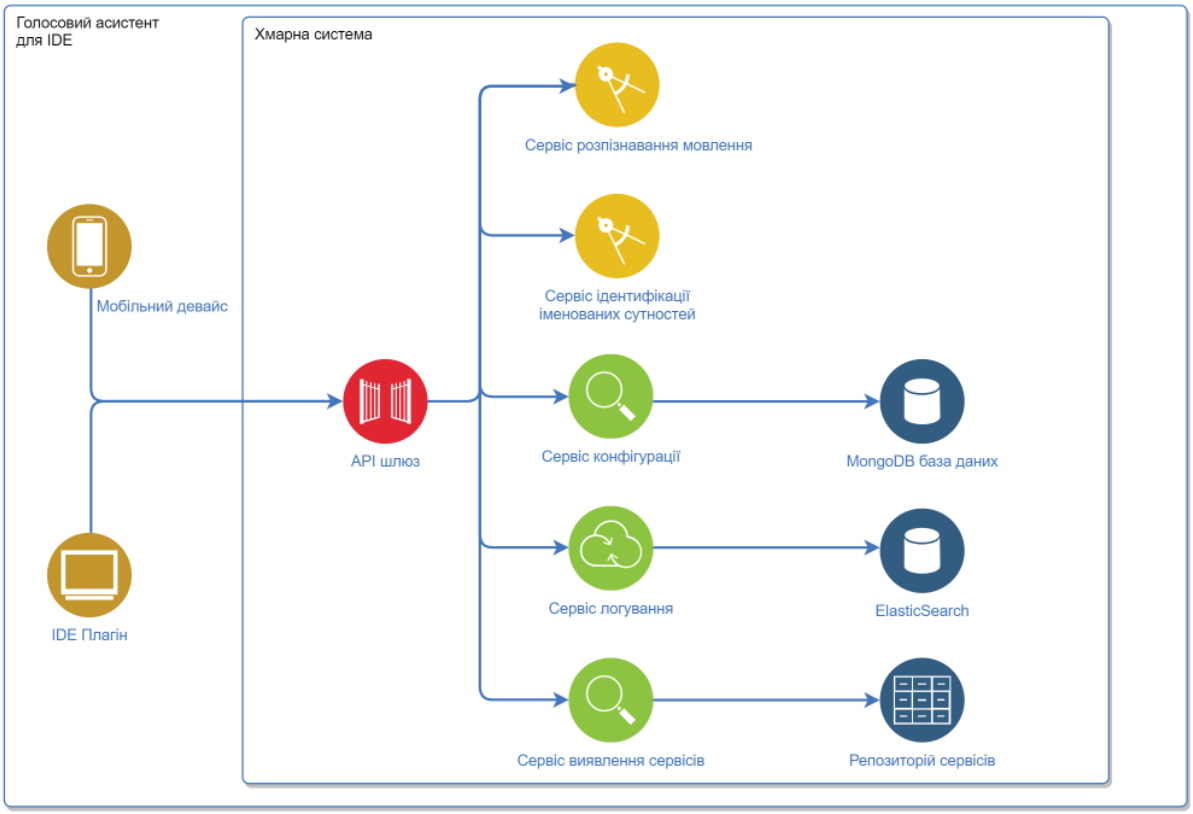


Рис. 12. Архітектура створеної системи

Проаналізувавши наведену архітектуру, можна помітити що користувачі взаємодіють безпосередньо з клієнтом (мобільним пристроєм або плагіном для IDE), який в свою чергу здійснює комунікацію з хмарним

сервером який і виконує всі важкі обчислення такі як розпізнавання команди користувача та ідентифікацію іменованих сутностей тексту програми. Кожен з сервісів має декілька екземплярів що забезпечує стійкість системи до помилок, адже якщо один із сервісів вийде з ладу – його замінить інший.

Аналогічно, завдяки створеній архітектурі системи, для того щоб додати підтримку нового інтегрованого середовища розроблення програмного забезпечення або нової мови програмування достатньо лише створити новий клієнт, який буде відповідати за виконання розпізнаних команд користувача.

Для забезпечення логування використовується ElasticSearch, а для збереження конфігурації використовується база даних MongoDB. Сервіс виявлення сервісів в свою чергу теж використовує репозиторій який містить інформацію про активні та доступні сервіси.

Детальний опис кожного із архітектурних компонентів (рис. 12) наведено нижче.

4.1.1. *Клієнт*

В програмній розробці для даної магістерської дисертації в ролі клієнта виступає плагін для обраного інтегрованого середовища розроблення (IntelliJ IDEA) та мови програмування Java. Проте, завдяки розмежування логіки клієнтської та серверної частин, в ролі клієнта може виступати мобільний пристрій або плагіни розроблені для інших середовищ та мов програмування. Зокрема, можуть бути створені плагіни для таких інтегрованих середовищ розроблення програмного забезпечення та текстових редакторів як Eclipse, NetBeans, PyCharm, Sublime, Microsoft Visual Code, CLion та інших.

IntelliJ IDEA – це інтегроване середовище розроблення програмних рішень, створене компанією JetBrains та підтримує велику кількість мов програмування, а саме Python, Java, JavaScript. Середовище дозволяє розробникам швидко реорганізовувати вихідні тексти програм та володіє

широким набором інструментів для рефакторингу. Візуальний інтерфейс системи зосереджений на підвищенні продуктивності розробників дозволяючи їм сконцентруватись на розробці функціональності. Даний продукт поставляється як у вигляді безкоштовної версії (Community) з урізаною функціональністю, так у вигляді преміум версії (Ultimate). Середовище дає можливість для розширення його функціональності шляхом створення користувацьких плагінів. Саме таким способом і була здійснена інтеграція розробленого голосового асистенту із обраним середовищем за допомогою клієнту.

При створенні плагіну для IntelliJ IDEA була використана мова програмування Java та SDK обраного інтегрованого середовища розроблення програмного забезпечення. Набір для розроблення програмного забезпечення (SDK) – це колекція засобів, які потрібні для розроблення застосування або плагіну для існуючого фреймворку або платформи. Наприклад, при розробленні програмного забезпечення з використанням Java необхідна Java SDK (JDK). SDK містять бінарні файли, вихідний текст програми для бінарних файлів та документацію для вихідного тексту програми. Як правило, SDK є глобальними. Це означає, що один SDK може використовуватися в декількох проектах і модулях.

Клієнт відповідає за отримання голосової команди від користувача, взаємодіє з інтегрованим середовищем розробки для побудови існуючих в контексті проекту доступних іменованих сутностей та відправлення їх через HTPP протокол з використанням архітектурного стилю REST на віддалений сервер для здійснення ресурсозатратних обчислювальних операцій. На хмарний сервер відправляються голосова команда користувача. Запис команди користувача відбувається з використанням Java Sound API.

Після отримання відповіді з серверу, клієнт знову взаємодіє з SDK інтегрованого середовища для виконання розпізнаної команди користувача. Система дозволяє користувачу за допомогою голосу створювати та змінювати змінні, методи, файли, класи, а також виконувати базові операції

з ними та обраним інтегрованим середовищем розроблення програмного забезпечення. Описаний алгоритм взаємодії клієнту системи з користувачем та інтегрованим середовищем наведений на діаграмі послідовностей на рис. 13.

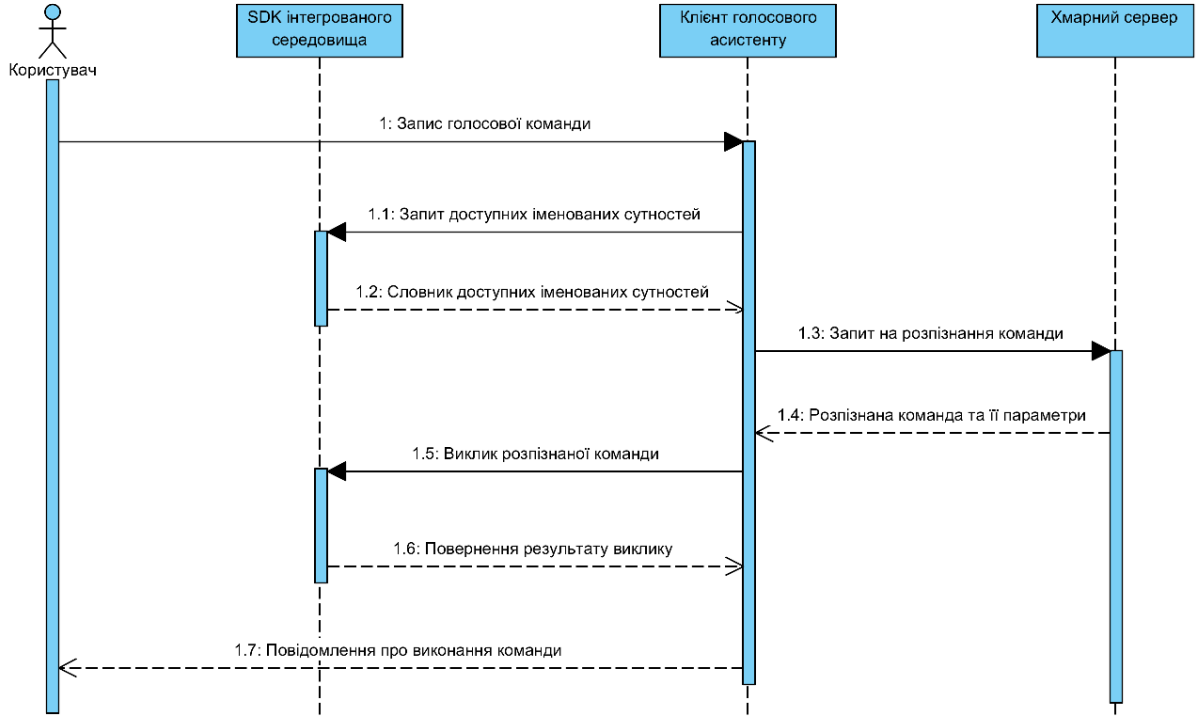


Рис. 13. Обробка голосової команди користувача

Уточнення розпізнаної команди користувача також відбувається на стороні клієнта, шляхом використання словника існуючих іменованих сутностей за допомогою модифікованого алгоритму наведеного в підрозділі 3.2. Це дозволяє уникнути лишнього навантаження на мережу та вирішити проблеми з приватністю, оскільки вся інформація про текст програми користувача залишається на його локальній машині та не передається на зовнішні сервіси.

4.1.2. *API шлюз*

API шлюз відповідає за агрегацію отриманих даних, а також за маршрутизацію запитів до сервісів системи. Він скаладає проміжний рівень між мікросервісами та клієнтами системи. Будь-які запити від користувачів

або інших систем спочатку надходять до API шлюзу, який після цього перенаправляє їх до декількох або одного конкретного мікросервісу. Результати, отримані від сервісів агрегуються і надсилаються клієнту у відповідь.

Теоретично, клієнт міг би напряму кидати запити до будь-якого мікросервісу, але такий підхід вніс би певні обмеження та проблеми в архітектуру системи. Однією з проблем що можуть виникнути є невідповідність інформації яку надає кожний мікросервіс та потребам клієнта. Залежно від розмірів системи, для отримання бажаних даних, що цікавлять клієнта, йому потрібно здійснити велику кількість викликів до різних мікросервісів. В такому випадку комбінування даних йому доведеться здійснювати самостійно, саме тому такий підхід значно ускладнює текст програми зі сторони клієнта. Систему стає складніше змінювати, адже напряму спілкується з сервісами.

Великою перевагою підходу із застосуванням API шлюзу є інкапсуляція внутрішньої структури системи. Замість того, щоб здійснювати виклики до конкретних сервісів, клієнт виконує запити лише до АПІ шлюзу. В такому випадку текст програми на стороні клієнта виглядатиме значно простішим.

В зв’язку із названими причинами, в даній магістерській дисертації було обрано підхід з використанням API шлюзу замість прямої взаємодії клієнта з сервісами.

Для імплементації API шлюзу застосовано Spring Cloud Gateway, який є прикладом реалізації Gateway паттерну в середовищі мікросервісів. Він використовує можливості серверу Netty, для того щоб переадресовувати запити до внутрішніх сервісів. Spring шлюз є неблокуючим, оскільки побудований поверх Spring WebFlux.

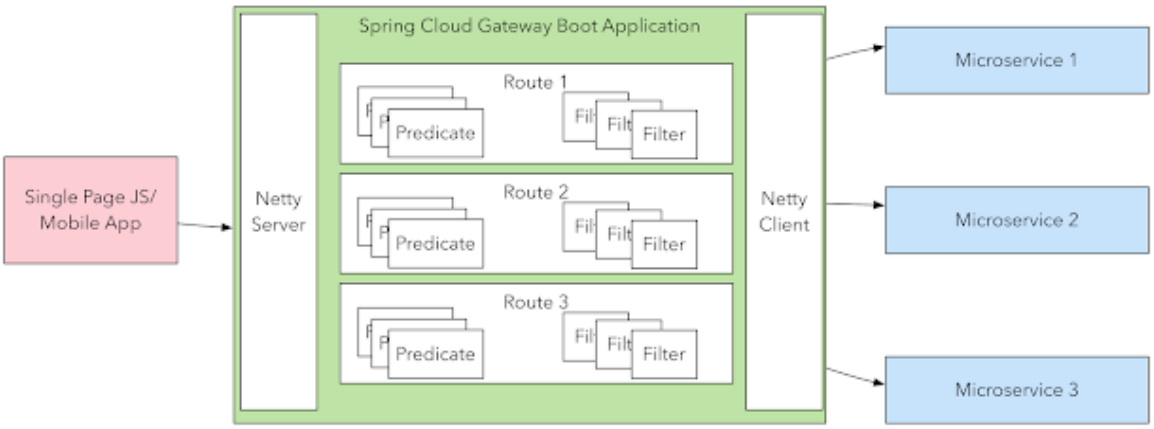


Рис. 14. Використання Spring API шлюзу

Spring шлюз оброблює деякі запити просто переадресовуючи їх до конкретного мікросервісу, а для інших запитів – звертається до багатьох сервісів, що містять дані та об'єднує отримані з них результати. Також, API шлюз здійснює аутентифікацію користувачів за допомогою сервісу авторизації та аутентифікації перед тим, як переадресувати запит до інших внутрішніх сервісів.

З метою покращення роботи системи вцілому, було додано на рівню шлюзу кешування запитів користувачів за допомогою бази даних Redis та Spring Caching. Для збору метрик та моніторингу стану системи використано Spring Actuator, який доповнює API інтерфейс шлюзу технічними характеристиками. Redis являє собою NoSQL систему керування базами даних яка працює зі структурами типу «ключ-значення». Використовується як для кешування та реалізації брокерів повідомлень, так як і база даних.

4.1.3. *Сервіс розпізнавання мовлення*

Сервіс розпізнавання мовлення призначений для перетворення голосової команди користувача в текст. В створеній архітектурі для голосового асистента, цей сервіс виконується на хмарному сервері і комунікація з ним відбувається через API шлюз.

Систему розпізнавання мовлення досить складно реалізувати оскільки існує дуже багато джерел варіативності, які впливають на точність розпізнавання речення. Нижче наведені найважливіші фактори, які повинні враховуватись при побудові сервісу для розпізнавання мовлення користувачів.

Насамперед, такою особливістю є стиль мовлення користувача. Кожна людина має різноманітний стиль ведення розмови, включаючи також різні акценти. Наприклад для англійської мови такими акцентами є американська англійська, британська англійська, австралійська англійська та багато інших, адже вона є найпоширенішою мовою у світі. Вимова конкретних слів або звуків також ускладнює для системи задачу розпізнавання мовлення.

Навколишнє середовище також впливає на точність розпізнавання, оскільки впливає на рівень фонового шуму під час записуванні команди. Запис голосової команди створений в ізольованій кімнаті буде містити набагато більше шумів порівняно з аудиторією, оскільки навіть відлуння може вносити шуми в систему. Також потрібно враховувати те, що кожна людина має власні характеристики мовлення людини, які залежать від багатьох факторів, включаючи тембр голосу та чіткість. Система розпізнавання мовлення потрібна враховувати та вирішувати вищезгадані проблеми для досягнення достатнього рівня точності.

Мова програмування Python містить багато модулів та бібліотек, які вирішують задачу розпізнавання мовлення та його конвертації в текст, зокрема – ApiAI, SpeechRecognition, Google Speech Cloud, AssemblyAI, PocketSphinx, Watson Developer Cloud, WIT та інші. Для реалізації сервісу розпізнавання мовлення використано мову програмування Python та модуль SpeechRecognition. Цей модуль надає широкий набір засобів від різних постачальників для розпізнавання мовлення та конвертації його в текст. Сервіс розпізнавання мовлення використовує модель «Speech to Text», від компанії IBM. Рішення що використовується побудоване на основі методів

глибокого машинного навчання, а саме з використанням рекурентної нейронної мережі із LSTM блоком та WaveNet моделей. Використане рішення надає можливість розпізнавати голосові команди на багатьох мовах і аудіо форматах.

Як результат сервіс розпізнавання мовлення повертає JSON відповідь з розпізнаним контентом в UTF-8 кодуванні через HTTP REST протокол.

4.1.4. *Сервіс ідентифікації іменованих сутностей*

Сервіс ідентифікації іменованих сутностей відповідає за розпізнавання команди користувача та її параметрів. Для реалізації розпізнавання використана рекурентна нейронна LSTM мережа та бібліотека SpaCy, яка реалізована на мові програмування Python.

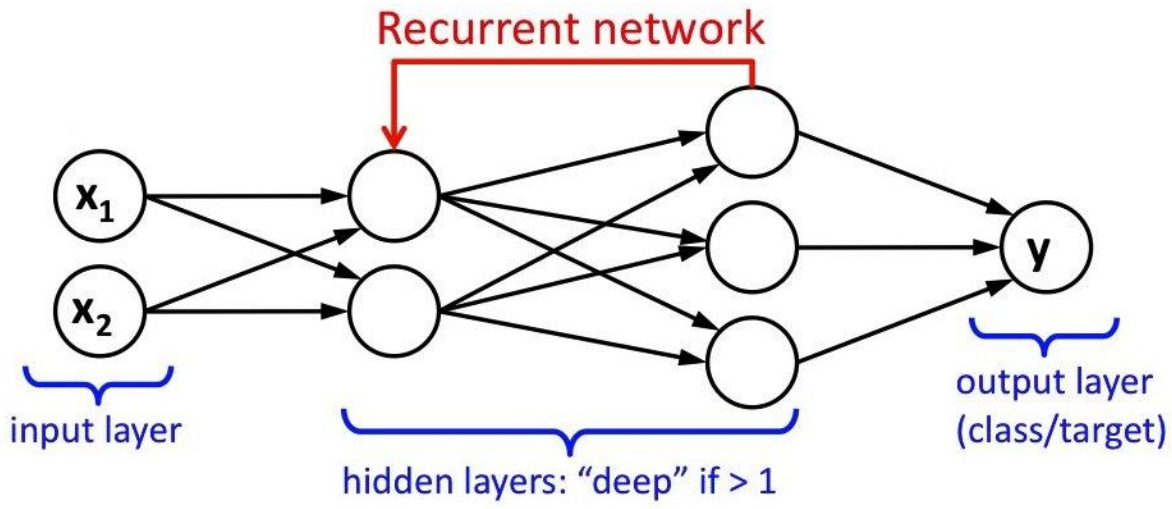


Рис. 15. Рекурентна нейронна мережа

SpaCy є популярною бібліотекою для аналізу текстів та обробки природньої мови на мові Python. Вона розроблена на базі мови програмування Cython, яка є підвидом Python проте із значно кращою швидкодією. Саме тому багато компаній віддають їй перевагу при реалізації власних проектів.

Бібліотека містить засоби для токенізації та класифікації текстів, визначення частин мови для слів, визначення залежності між словами в реченні, розпізнавання іменованих сутностей, а інтеграцію із засобами глибокого машинного навчання і візуалізацію даних.

Сервіс використовує HTTP протокол та REST підхід для передачі повідомлень. Кожен вхідний запит і розпізнанні іменовані сутності логуються за допомогою сервісу журналювання подій. Таким чином можна відслідкувати, які іменовані сутності були для конкретної команди користувача.

4.1.5. *Сервіс конфігурації*

Сервіс конфігурацій являє собою централізоване місце для керування зовнішніми налаштуваннями для кожного сервісу незалежно від середовища. Spring Cloud Config було обрано як його реалізацію, яка спрощує управління зовнішньої конфігурації для великих розподілених систем як на стороні сервера, так і зі сторони клієнта.

До того, як мікросервіси стали поширеними, файли конфігурації створювалися таким чином, що в них вносили зміни, то потрібно перезавантажувати контейнер. Зазвичай, при розробці системи застосовується підхід із декількома різними середовищами, для кожного з яких необхідно підтримувати свій файл конфігурації, а для того щоб переключитись між ними, необхідно призупиняти роботу всієї системи. Тому, дуже важливою проблемою є те, що кодова база тісно пов’язана з файлами конфігурації, і відповідно, будь-які зміни що вносились спричиняють розгортання та збирання всієї системи знову.

Всі згадані вище проблеми вирішує сервіс хмарної конфігурації.

Перевагою такого підходу із централізованою конфігурацією є те, що у тому випадку коли до налаштувань мікросервісу вносяться зміни, то вони будуть оброблені на льоту, і відповідно без потреби збирати цей мікросервіс знову.

Всі налаштування сервіс конфігурації зберігає в SVN(системі контролю версій), до якої під час роботи системи можуть вноситись зміни.

Мікросервіси, якщо їм необхідна та чи інша конфігурація, роблять виклик до нього за допомогою REST API. Схема роботи названого сервісу показана на рис. 16.

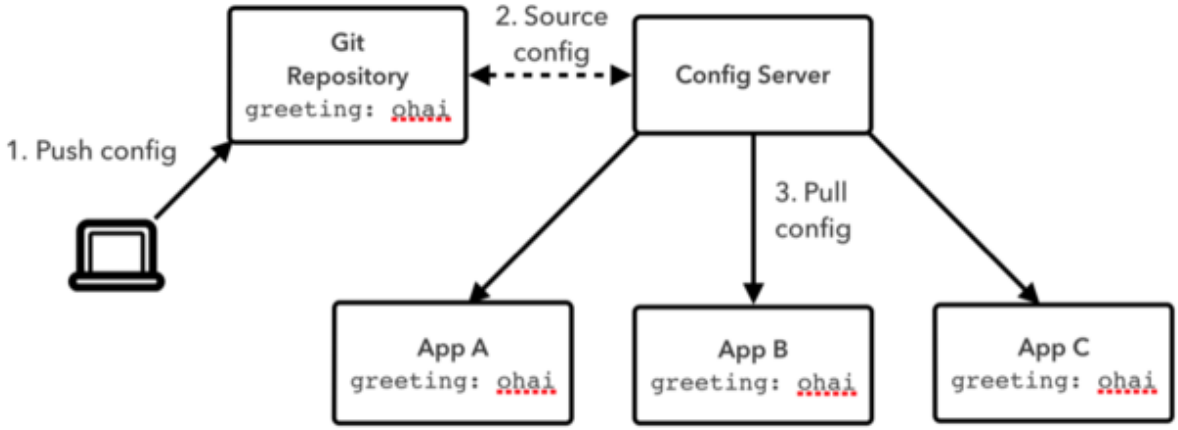


Рис. 16. Принцип роботи сервісу конфігурацій

4.1.6. *Сервіс виявлення сервісів*

Використовуючи мікросервісну архітектуру, сервіс ідентифікації сервісів займає важливе місце, адже, кожен сервіс існує в багатьох екземплярах тому необхідний механізм для ідентифікації адрес інших сервісів для їх подальших викликів, без явного введення хосту і порта для кожного мікросервісу до виконується звернення. Тим більше, під час роботи системи, адреси мікросервісів можуть змінюватись в реальному часі. Наприклад, можуть створюватись нові екземпляри мікросервісів, або виходити з ладу старі. Тому механізм автоматичної реєстрації є дуже важливим для виявлення сервісів. При реалізації сервісу виявлення та реєстрації сервісів було використано Netflix Eureka та Spring Cloud рішення.

Netflix Eureka застосовується для того, щоб побудувати реєстр мікросервісів та виконувати автоматичний запис кожного мікросервісу до нього при запуску. Після цього всі інші сервіси мають можливість

викликати його, звертаючись з використанням ідентифікатору сервісу, який був введений при реєстрації та є унікальним.

Spring Cloud використовується для спрощення виявлення сервісів та створення реєстру, а також використовує Ribbon в якості балансувальника. Він здійснює балансування запитів на сервер ще на клієнті. Перед тим як здійснити запит, клієнту потрібно звернутися до мікросервісу ідентифікації сервісів, щоб дізнатись потрібну інформацію, таку як імена хостів, IP-адреси, типи портів, тощо. Після цього Ribbon з побудованого списку, застосовуючи Round-Robin, здійснює вибір одного екземпляру до якого і виконується запит. Шоб не виконувати запит до сервісу ідентифікації сервісів кожного разу, дані отримані від нього записуються в кеш для кожного клієнта. Метод розподілення запитів зі сторони клієнта наведена на рис. 17.

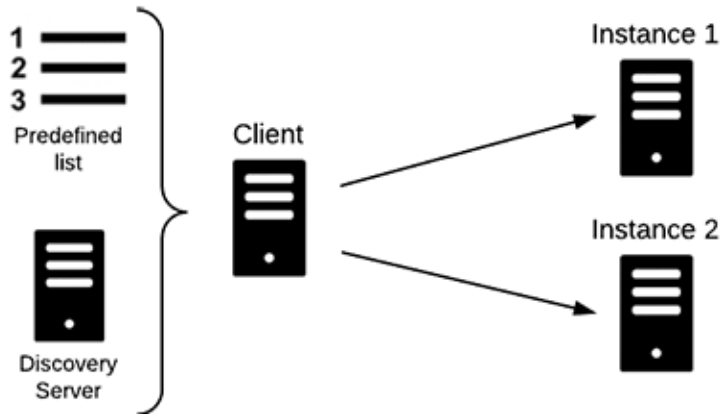


Рис. 17. Схема балансування на стороні клієнта

Перевагами балансування на стороні клієнта є стійкість та децентралізація, тому саме такий підхід було обрано для реалізації в даному дипломному проекті.

4.1.7. *Сервіс логування*

Оскільки всі сервіси працюють на різних портах та хостах, необхідне централізоване, зовнішнє місце де буде зберігатись інформація про події, що виникають у роботі голосового асистенту. Розподілена система

журналювання дає змогу переглядати та аналізувати інформацію про всі події, які відбуваються у багатьох різних мікросервісів системи та спрощує процес відладки та пошуку помилок.

Сервіс використовує HTTP протокол та REST підхід для передачі повідомлень. Кожен вхідний запит і розпізнанні іменовані сутності логуються за допомогою сервісу журналювання подій. Таким чином можна відслідкувати, які іменовані сутності були для конкретної команди користувача.

Для вирішення проблеми журналювання подій обрано ELK набір технологій, а саме Logstash, ElasticSearch і Kibana.

- Filebeat + Logstash – це динамічний конвертер інформації з широкою системою доповнень та потужною взаємодією з ElasticSearch. Внаслідок того, що Filebeat збирає файли з записаними подіями з кожного сервісу системи та записує їх до ElasticSearch сховища, його необхідно встановлювати на всіх серверах, з екземплярами мікросервісів. Filebeat є програмою, єдиним завданням якої є відправлення журналів з кожного сервісу до системи Logstash, що надає змогу встановлення компнента Logstash лише в єдиному місці. Вона має невеликий розмір та використовує мало ресурсів системи.
- ElasticSearch є пошуковою, розподіленою та аналітичною системою, яка створена з метою максимальної надійності, горизонтального масштабування і простого управління інформацією.
- Kibana застосовується для надання візуалізації даних через інтерфейс користувача.

На рис. 18 наведено принципи комунікації компонентів ELK набору один з одним. Кожний сервіс логвє події, що виникають в процесі його роботи в свій певний файл. Filebeat, який є встановленим на кожному сервері де потрібно здійснювати збір даних для їх подальшого

перенаправлення, передає дані з цих файлів до компонента Logstash. Він, в свою чергу, отримує передані дані, конвертує їх в формат JSON та надсилає далі до ElasticSearch для їх подальшого зберігання. Kibana отримує доступ до записів в ElasticSearch та виконує візуалізацію для клієнта.

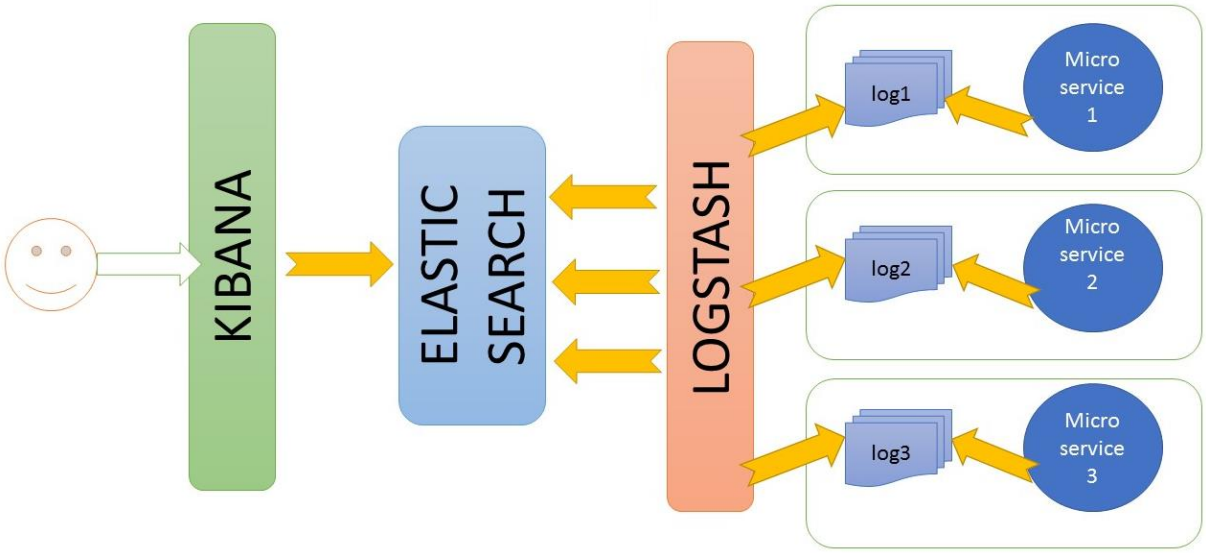


Рис. 18. ELK стек для логування

4.2. Тестування розробленого програмного забезпечення

Завдячуючи розподіленій природі коду, тестування мікросервісів стає набагато складнішим за тестування цілісної системи. Зважаючи на те, що я мав глибоке уявлення про систему та з метою найбільшого покриття тестами, я використовував метод білого ящика.

Використовувалися юніт тести для роздільного тестування кожного мікросервісу. Це робилося з метою перевірки того чи не приведуть до регресії нові зміни в коді. Для тестування коду клієнтської частини використовувалася бібліотека Jasmine, а JUnit та Mockito фреймворки для серверної частини з ціллю реалізації модульного тестування. На ранніх етапах циклу розроблення програмного забезпечення юніт тести знаходять найбільше проблем. Вони містять як похибки при імплементації проекту так і «загублені» складові специфікації для певного модулю або частини системи. Ціна ідентифікованого багу на ранніх етапах розробки і тестування

є значно нижчою порівняно з вартістю його виявлення та виправлення на подальших етапах розробки програмного забезпечення. Крім того, модульні тести значно спрощують інтеграцію з іншими системами, адже можна впевнитись що всі модулі в ізоляції працюють згідно реалізації розробника.

Для виконання «смок-тестів» або так-званого димового тестування, а також покриття критичного шляху системи використано фреймворк Selenium. На даному етапі тестування виконано перевірку всіх основних функції системи згідно поставлених вимог.

Створені тести дають можливість підвищити якість реалізованого програмного забезпечення та отримати впевненість в тому, що введення нових змін не спричинить помилок в існуючій реалізації.

4.3. Аналіз розробленої системи

Розроблене програмне рішення для голосового асистенту для інтегрованих середовищ розроблення програмного забезпечення було створене з використанням мікросервісної архітектури. Система складається з клієнта, API шлюзу та 7 сервісів, кожен з яких відповідає за свою частину застосування.

Клієнт представлений у вигляді плагіну для IntelliJ IDEA, проте аналогічні клієнти можна створити для інших інтегрованих середовищ розроблення програмного забезпечення, так і для IDE на мобільних пристроях.

Система розпізнає команди користувача англійською мовою та надає можливість створення та редагування змінних, функцій, класів, файлів, а також керуванням середовищем розроблення. Завдяки використанню модифікованого алгоритму розпізнавання іменованих сутностей, який враховує контекст користувача – точність розпізнавання команд є досить високою.

Серед недоліків системи можна виділити те, що для використання її повної функціональності необхідний доступ до мережі, оскільки всі

обчислення з розпізнавання команди користувача та іменованих сутностей виконуються на сервері. Також доцільно додати підтримку інших мов, окрім англійської.

Створений голосовий асистент для інтегрованих середовищ розроблення дозволяє підвищити продуктивність розробників, надає можливість написання тексту програми на мобільних пристроях, а також для людей з обмеженими можливостями.

Голосовий асистент задовольняє всім поставленим в розділі 1 критеріям оцінювання, а саме він є орієнтованим та спеціально створеним для інтегрованих середовищ розроблення програмного забезпечення, дозволяє задавати користувачам команди високого рівня звичайною розмовною мовою, є безкоштовним у використання, містить відкритий текст програми та спроектований з використанням мікросервісної архітектури, що дозволяє легко вносити зміни в систему.

4.4. Рекомендації щодо подальшого вдосконалення

Для подальшого вдосконалення системи доцільно розширити список підтримуваних інтегрованих середовищ розроблення програмного забезпечення, мов програмування та команд користувача шляхом введення нових клієнтів. Завдяки тому, що створена архітектура застосування не має тісної прив’язки до конкретного клієнту і спрощує досягнення крос-платформності, тому доцільним буде реалізація клієнту на базі мобільних платформ.

Оскільки зараз створений голосовий асистент може повноцінно функціонувати лише в онлайн режимі (з наявним підключенням до мережі інтернет), тому доречно, хоча і з певною втратою в точності розпізнавання, розширити клієнт шляхом використання менш ресурсозатратних алгоритмів для розпізнавання команд користувача та ідентифікації іменованих сутностей. Підвищення точності розпізнавання можна досягти шляхом усунення шумів на етапі конвертації голосового повідомлення користувача

в текст. Оскільки зараз система підтримує лише команди задані англійською мовою, тому доцільно розширити її, шляхом підтримки інших мов для введення команд.

Незважаючи на вказані зауваження, система є повністю готовою до використання.

4.5. Висновки

У даному розділі розроблено архітектуру системи голосового асистенту для інтегрованих середовищ розроблення програмного забезпечення. Наведено детальний опис кожного із складових компонентів системи – мікросервісів. Описано процес тестування системи та шляхи для її подальшого вдосконалення. Проведено аналіз відповідності створеної системи поставленим критеріям.

5. ПОБУДОВА БІЗНЕС МОДЕЛІ

5.1. Аналіз існуючих проблем

Області машинного навчання та обробки природної мови набули значного розвитку протягом останнього десятиліття. Оскільки в згаданих сферах використовуються статистичні методи та підходи, що дають змогу алгоритмам виділяти та отримувати нову інформацію на основі існуючих даних, то тенденція до стрімкого зростання цих галузей легко пояснюється значним збільшенням обсягів наявної інформації. Результати останніх досліджень в областях машинного навчання та обробки мовлення швидко знаходять застосування в сферах медицини, освіти, логістиці та інших.

Сфера інформаційних технологій теж знаходиться в стані постійного розвитку та змін – створюються нові мови програмування, фреймворки та бібліотеки, пропонуються нові ідеї, засоби та підходи до розроблення програмного забезпечення. Надзвичайно важливу роль для розробників відіграють ті засоби, які використовуються ними в процесі написання тексту програми – інтегровані середовища розроблення програмного забезпечення. Незважаючи на те, що вони значною мірою полегшують життя розробників наявністю таких корисних компонентів та функцій як автодоповнення тексту програми, зручного редактора тексту програми, засобів для автоматизації збірки проекту та відлагодження програм – все таки в цій області все ще існують проблеми, які потребують вирішення.

Першою проблемою є відсутність можливості написання тексту програми для людей з обмеженими можливостями. Також багато розробників, в силу набутих чи вроджених травм, втрачають доступ до розроблення програмного забезпечення і фактично не мають змоги займатись справою яка їм цікава або приносить засоби для існування.

Наступним недоліком є те, що звичайні засоби комунікації з комп’ютером такі як клавіатура чи мишка не є ефективними в плані швидкості введення інформації і відповідно впливають на продуктивність

розробника. Вирішення згаданої проблеми також дасть змогу програмісту концентруватись на вирішенні поставленої задачі, а не на деталях її реалізації.

Ще однією проблемою є неможливість написання тексту програми за допомогою мобільних пристроїв. Сьогодні, враховуючи особливості наведених платформ, використання інтегрованих середовищ розроблення на них є малопродуктивним або взагалі неможливим, оскільки швидкість введення інформації розробником є досить низькою та заставляє бажати кращого.

Вище описані проблеми узагальнені у вигляді «Дерева проблем», на рис. 19.

Рис. 19. Дерево проблем

Жоден з існуючих програмних засобів не вирішує в повній мірі наведених проблем. Детальний аналіз та порівняння таких систем наведений в розділі 1. Основним їх недоліком є те, що вони не дають можливості користувачам керувати середовищем розроблення програмного забезпечення та процесом написання тексту програми за допомогою звичних мовних конструкцій, а потребують від них вивчення створеної ними системи команд та скорочень для вирішення цих задач. Такий підхід потребує від клієнтів значних витрат часу та довгий період налаштувань та адаптації перед тим як розпочати роботу з системою.

5.2. Визначення зацікавлених сторін

У реалізації рішення для вищеописаних проблем зацікавлені декілька сторін.

Найбільш очевидною зацікавленою особою є власне самі розробники програмного забезпечення, оскільки продукт направлений на вирішення саме їх проблем. Реалізація продукту значною мірою підвищить продуктивність вказаної групи осіб, відкриє нові можливості такі як можливість створення програмного забезпечення на нових платформах, дозволить концентруватись на вирішенні задачі а не на реалізації програмного рішення для неї. Для розробників з обмеженими можливостями це здатність займатись програмуванням та отримувати від цього задоволення, оскільки вони власними руками втілюватимуть свої ідеї в життя, або матеріальну винагороду.

Іншою важливою та впливовою зацікавленою стороною є керівники IT-відділів компаній, що займаються розробкою нового або підтримкою існуючого програмного забезпечення. Цінність продукту для них полягає в тому, що він підвищить продуктивність не лише конкретної особи, а й відділу в цілому, що дасть їм змогу в менший проміжок часу виконувати поставлені цілі та завдання.

Останньою, але не менш важливою, групою зацікавлених осіб є власники існуючих інтегрованих середовищ розроблення програмного забезпечення. Оскільки конкуренція на цьому ринку є досить високою, то інтеграція з рішенням, описаним в даній магістерській дипломній роботі, дасть їм змогу збільшити кількість користувачів їх продукту і відповідно зайняти більшу частину ринку.

На даному етапі сформована стратегія приваблення зацікавлених сторін полягатиме у проведенні спеціальних презентацій розробленого продукту для їх представників, участь у спеціалізованих конференціях, форумах та виставках.

У табл. 4 наведено вищезгадані групи зацікавлених сторін, їх вплив (міра зацікавленості у вирішенні наявних проблем) та інтереси втілення продукту в життя.

Таблиця 4

Аналіз зацікавлених сторін

Зацікавлена сторона	**Інтерес зацікавленої особи**	**Вплив**	**Стратегії приваблення зацікавлених сторін**
Розробники програмного забезпечення	Підвищення продуктивності роботи, можливість розроблення програмного забезпечення на нових платформах. Інтерес: високий	Високий	Проведення презентацій розробленого продукту для представників зацікавлених сторін. Публікація інформації про створений продукт на спеціалізованих інформаційних ресурсах.
Керівники ІТ-відділів	Підвищення продуктивності роботи команди. Інтерес: середній	Середній	
Власники існуючих інтегрованих середовищ розроблення	Збільшення кількості користувачів їх продуктів, залучення нових клієнтів Інтерес: високий	Середній	

5.3. Характеристика комерційного рішення

Кінцевий продукт повинен вирішувати всі вище зазначені існуючі проблеми. Даний програмний продукт буде реалізовувати, описані у попередній розділах, алгоритми та підходи до обробки голосових повідомлень, конвертація мовлення в текст, визначення іменованих сутностей в тексті, інтерпретації команд користувача та їх виконання.

Рішення надаватиме засоби інтеграції з найбільш популярними середовищами розроблення програмного забезпечення та мовами програмування.

Втілення даного продукту в життя шляхом створення голосового асистенту для інтегрованих середовищ розроблення програмного забезпечення вирішить вищезгадані проблеми та надасть зацікавленим особам наступні переваги:

- Надасть можливість написання тексту програми для людей з обмеженими можливостями. Багато людей, в силу набутих чи вроджених травм, втрачають доступ до розроблення програмного забезпечення і фактично не мають змоги займатись справою яка їм цікава або приносить засоби для існування.
- Підвищить продуктивність розробників, оскільки голос є найбільш ефективним засобом комунікації. Абстрагування від звичайних засобів введення інформації таких як мишка чи клавіатура, дасть змогу програмісту в ефективніший спосіб використовувати власні можливості та можливості комп’ютера. Такий підхід дозволить розробникам виконувати складні дії лише за допомогою однієї розмовної фрази, концентруватись на вирішенні задачі, а не на деталях її реалізації.
- Дозволить написання тексту програми за допомогою мобільних пристроїв. Сьогодні, враховуючи особливості наведених платформ, використання інтегрованих середовищ розроблення на них є малопродуктивним або взагалі неможливим, оскільки швидкість введення інформації розробником є досить низькою та потребує покращення.

Розроблене комерційне рішення поставлених проблем повинне відповідати всім поставленим обов’язковим вимогам до програмного забезпечення, які наведені в розділі 2.

5.4. Конкурентні переваги рішення

Так як вже зазначалося, що області машинного навчання та обробки природної мови розпочали показувати хороші результати лише в останні роки, то реалізацій рішення поставлених проблем з використанням підходів та методів з цих галузей зараз дуже мало. Існуючі рішення мають ряд недоліків, які були описані та проаналізовані в розділі 1 даної магістерської дисертаційної роботи.

Всі недоліки, що мають аналоги, вирішуються за допомогою розробленого програмного продукту з використанням сучасних підходів та методів машинного навчання, а також покращених алгоритмів для їх застосування в заданій предметній області.

Отже, основними конкурентними перевагами створеного програмного рішення є:

- Можливість управління інтегрованим середовищем розроблення програмного забезпечення та процесом написання тексту програми за допомогою звичних мовленнєвих конструкцій.
- Модульність створеної системи, що дозволяє легко інтегруватись з будь-якою частиною розробленого рішення.
- Інтеграція з основними середовищами розроблення програмного забезпечення та мовами програмування.
- Використання покращених методів, які враховують специфіку обраної доменної області, що дозволяє давати більшу точність розпізнавання команд та іменованих сутностей порівняно з існуючими аналогами.
- Низький рівень входу для нових клієнтів.
- Порівняно невисока вартість продукту.
- Спрощення процесу написання тексту програми, шляхом його генерації на основі заданих користувачем команд.

Можливість використання тимчасової підписки дозволить користувачеві спершу випробувати функціональність, що пропонується продуктом і на основі цього зробити рішення про подальше використання продукту та його покупку.

5.5. Аналіз сегментів ринку споживачів

Проведемо сегментацію ринку споживачів для досягнення максимальної ефективності діяльності команди проекту, тобто виділимо більш-менш однорідні групи користувачів, які є зацікавленими в розробці запропонованого продукту.

Як вже зазначалося вище, клієнтами даного програмного забезпечення є всі зацікавлені особи. Отже, сегментацію ринку клієнтів для голосового асистенту для інтегрованих середовищ розроблення програмного забезпечення здійснимо за двома наступними категоріями:

- платоспроможність клієнтів, тобто здатність купити ліцензію на використання продукту;
- обсяг запланованих робіт.

До першої групи користувачів відносяться клієнти, професійна діяльність яких, переважно, пов'язана з розробкою програмного забезпечення. Цей сегмент користувачів включає в себе клієнтів, які будуть основними покупцями ліцензій продукти і відповідно будуть основним джерелом доходу для компанії. Також до цієї категорії відносяться власники існуючих інтегрованих середовищ розроблення програмного забезпечення, які зацікавленні в розширенні функціональності своїх продуктів та збільшення власної клієнтської бази.

До другої категорії, відповідно, належать користувачі, які лише знайомляться або навчаються розробці програмного забезпечення, або не мають достатніх коштів для оплати ліцензії. В основному до цієї категорії належать студенти та розробники-початківці.

Отже, для співпраці, найбільш перспективними є споживачі, що належать до першого сегменту користувачів: розробники, керівники IT-відділів, власники інтегрованих середовищ розроблення. Саме платоспроможні клієнти є найбільш привабливими для бізнесу та рекламодавців.

5.6. Унікальна ціннісна пропозиція

Продукт, що пропонується клієнтам – це голосовий асистент для управління інтегрованими середовищами розроблення програмного забезпечення та процесом написання тексту програми. У підрозділі 5.1 було здійснено аналіз існуючих проблем та представлено їх у вигляді дереві проблем, а для зацікавлених сторін – було визначено їх очікування щодо того, які функції повинен надавати продукт, щоб задовільнити потреби та зацікавити потенційних клієнтів.

Основною унікальною ціннісною пропозицією є розроблена система для голосового управління інтегрованим середовищем розроблення з використання покращених методів, які враховують специфіку обраної доменної області.

5.7. Аналіз доходів та витрат

Основними складовими доходу компанії будуть продаж ліцензії на використання програмного забезпечення, а також надання консультацій та послуг щодо його підтримки. Продаж програмного забезпечення планується шляхом укладання угоди, яка надає право використовувати програмне забезпечення, тобто продажом ліцензії на використання даного програмного забезпечення. Платна технічна підтримка складатиме обробку запитів споживачів та усунення недоліків роботи системи в терміни, які закладені в договорі про використання програмного забезпечення.

Планується надавати користувачам декілька видів ліцензій, щоб охопити клієнтів які належать до різних сегментів користувачів.

Корпоративна ліцензія включати повну функціональність системи. Оплата буде здійснюватися за підпискою та буде розподілена на певні часові проміжки використання. Вартість підписки на один часовий проміжок становитиме 10$. Основними користувачами такої ліцензії будуть розробники програмного забезпечення, яким потрібен даний продукт для використання в професійних цілях.

Наступний вид ліцензії, яка буде запропонована – це ліцензії, яка не призначена для використання в корпоративних цілях. Вона не буде містити повного набору функцій, які доступні в корпоративній версії. Основними її користувачами будуть клієнти із другого сегменту споживачів, тобто студенти та розробники-початківці які не готові заплатити кошти за версію з повною функціональністю.

Останній вид ліцензії – це пробна ліцензія, яка призначена для ознайомлення користувачів з продуктом та є обмеженою в часі. Вона буде надавати повний набір функціональності системи.

До витрат на розробку та підтримку програмного рішення відноситься наступне:

- оплата праці персоналу;
- оренда приміщення;
- оплата податків;
- витрати на необхідне програмне забезпечення та обладнання;
- оплата послуг юриста.

Всі наведені витрати на проект доцільно розбити на дві категорії: одноразові витрати та постійні. Одноразові витрати – це ті кошти, які вкладаються при старті проекту, а постійні – вкладаються в розвиток проекту кожного місяця.

Прогнозовані одноразові та постійні витрати на реалізацію продукту наведені в табл. 5. На основі проведеного аналізу можна помітити, що для старту проекту необхідно 9500$ і в подальшому 9200$ витрат кожного місяця.

Таблиця 5

Аналіз прогнозованих витрат

Одноразові витрати		**Щомісячні витрати**	
Найменування витрат	**Витрати, доларів**	**Найменування витрат**	**Витрати, доларів**
Юридичні послуги	4000	Зарплата працівникам	2000 * 3
Програмне забезпечення та технічне обладнання	5000	Оренда приміщення	3000
Транспортні витрати	500	Оплата додаткових послуг (інтернет, комунальні послуги)	200
Сумарні витрати	9500	Сумарні витрати	9200

Як вже було зазначено раніше основним джерелом доходу є продаж ліцензій на користування продуктом. Враховуючи, що вартість місячної підписки становитиме 10 доларів, то після релізу продукту очікується середній об’єм продаж в розмірі 1500 ліцензій і відповідно щомісячний дохід в розмірі 15000$.

Щомісячний прибуток після релізу продукту обраховується як різниця доходу та щомісячних витрат і становитиме: 15000 – 9200 = 5800$. Отже, на основі аналізу можна помітити що продукт має перспективи для реалізації з фінансової точки зору.

5.8. Характеристика узагальненої бізнес-моделі

Підсумуємо вищенаведену інформацію про продукт у вигляді лаконічної lean canvas бізнес-моделі.

Проблема: відсутність можливості написання тексту програми для людей з обмеженими можливостями; неможливість написання тексту програми з використанням мобільних пристроїв; звичайні засоби комунікації з комп'ютером такі як клавіатура чи мишка не є ефективними в плані швидкості введення інформації і відповідно впливають на продуктивність розробника.

Рішення: програмне забезпечення, що дозволяє користувачу керувати інтегрованим середовищем розроблення і процесом написання тексту програми за допомогою голосу.

Споживачі: розробники програмного забезпечення, керівники IT-відділів, власники інтегрованих середовищ розроблення програмного забезпечення.

Унікальна ціннісна пропозиція: розроблена система для голосового управління інтегрованим середовищем розроблення з використання покращених методів та алгоритмів, які враховують специфіку даної доменної області, що дозволяє давати більшу точність розпізнавання команд та іменованих сутностей порівняно з існуючими аналогами.

Потоки доходів: доходи від продажу ліцензій та підтримки програмного забезпечення.

Структура витрат: оплата праці персоналу; оренда приміщення; оплата податків; витрати на потрібне програмне забезпечення та обладнання; оплата послуг юриста.

Канали (шляхи до користувачів): інформаційні ресурси та форуми, відділи співпраці IT-компаній.

Ключові метрики: кількість проданих ліцензій.

Канва бізнес-модель побудована та представлена у зведеному вигляді в табл. 6.

Отже, проведений аналіз дозволяє говорити що розроблений голосовий асистент для інтегрованих середовищ розроблення програмного забезпечення має хороші перспективи для подальшого розвитку.

Таблиця 6

Канва бізнес-моделі

<table>
<tr>
<td>Проблема
відсутність можливості написання тексту програми для людей з обмеженими можливостями;

неможливість написання тексту програми з мобільних пристроїв;

неефективність звичайних засобів комунікації з комп’ютером</td>
<td>Рішення
програмне забезпечення, що дозволяє користувачу керувати інтегрованим середовищем розроблення і процесом написання тексту програми за допомогою голосу</td>
<td>Унікальна ціннісна пропозиція
розроблена система для голосового управління інтегрованим середовищем розроблення</td>
<td>Канали
інформаційні ресурси та форуми, відділи співпраці ІТ-компаній.</td>
<td>Споживачі
розробники програмного забезпечення;
керівники ІТ-відділів;
власники інтегрованих середовищ розроблення програмного забезпечення</td>
</tr>
<tr>
<td colspan="3">Структура витрат
оплата праці персоналу;
оренда приміщення;
оплата податків;
оплата послуг юриста</td>
<td colspan="2">Потоки доходів
доходи від продажу ліцензій та підтримки програмного забезпечення</td>
</tr>
</table>

5.9. Висновки

У даному розділі було проведено аналіз поточної ситуації у сфері розроблення програмного забезпечення, виявлено наявні проблеми та представленого їх у вигляді дерева проблем для зручнішого аналізу. Крім цього було проведено аналіз зацікавлених сторін, їх інтереси та ступінь впливу на реалізацію продукту. Внаслідок чого було запропоновано комерційне рішення з конкурентними перевагами, що задовольняє інтереси зацікавлених осіб, та сформовано унікальну ціннісну пропозицію запропонованого продукту. Розподіл потенційних клієнтів на сегменти користувачів дав змогу спрогнозувати потенційні доходи та витрати на реалізацію продукту. На основі отриманих даних було здійснено синтез бізнес-моделі, що дає змогу обґрунтувати доцільність реалізації даного продукту та прогнозує її потенційну прибутковість в майбутньому.

ВИСНОВКИ

У ході виконання даної магістерської дипломної дисертації розглянуто та удосконалено існуючі підходи до розпізнавання іменованих сутностей тексту програми шляхом врахування контекстної інформації для обраної предметної області. Запропоновано модифікований метод розпізнавання іменованих сутностей тексту програми, який полягає в уточненні результатів отриманих стандартними методами шляхом використання інформації про вже введені сутності в текст програми та забезпечує підвищення точності на 5%.

З використанням запропонованого модифікованого методу ідентифікації іменованих сутностей розроблено програмне рішення для голосового асистенту інтегрованих середовищ розроблення програмного забезпечення. Система побудована на основі мікросервісного підходу, що забезпечує масштабованість та невеликі витрати ресурсів при додаванні нової функціональності або заміни існуючих реалізацій. Система складається з клієнта, API шлюзу та 7 сервісів, кожен з яких відповідає за свою частину застосування. Клієнт представлений у вигляді плагіну для IntelliJ IDEA та дозволяє користувачам за допомогою голосових команд керувати процесом написання тексту програми, а також інтегрованим середовищем розроблення програмного забезпечення.

Створений голосовий асистент для інтегрованих середовищ розроблення дозволяє підвищити продуктивність розробників, надає можливість написання тексту програми на мобільних девайсах, а також для людей з обмеженими можливостями.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Hidden Markov Models for Speech Recognition [Електронний ресурс]. — Режим доступу: http://www.machine-models.com/markov.html. — Дата доступу : квітень. 2019
2. Монолітна архітектура розроблення ПЗ [Електронний ресурс]. — Дата візиту 21.02.2019. — Режим доступу до ресурсу: http://microservices.io/patterns/monolithic.html.
3. Документація мови програмування “Java” [Електронний ресурс]. — Дата візиту 21.02.2018. — Режим доступу до ресурсу: https://docs.oracle.com/javase/specs/jls/se9/html/index.html
4. Robust Speech Recognition Using Generative Adversarial Networks (2017), Anuroop Sriram et al., Baidu Research Group
5. Wav2Letter: an End-to-End ConvNet-based Speech Recognition System (2016), Ronan Collobert et al., Facebook AI Research
6. Long short-term memory recurrent neural network architectures for large scale acoustic modeling (2014), Hasim Sak et al., Google Research
7. Бібліотека “Stanford Core NLP” [Електронний ресурс]. — Дата візиту 13.05.2019. — Режим доступу до ресурсу: https://stanfordnlp.github.io/CoreNLP/index.html.
8. Документація фреймворку “Spring” [Електронний ресурс]. — Дата візиту 13.05.2019. — Режим доступу до ресурсу: https://spring.io .
9. Документація модулю “Spring Cloud Gateway” [Електронний ресурс]. — Дата візиту 11.04.2019. — Режим доступу до ресурсу: https://dzone.com/articles/spring-cloud-gateway-configuring-a-simple-route.
10. Документація бази даних “MongoDB” [Електронний ресурс]. — Дата візиту 11.04.2019. — Режим доступу до ресурсу: https://www.mongodb.com/.
11. Сахарчук Т.Ю. Модифікований метод розпізнавання іменованих сутностей тексту програми / Т.Ю. Сахарчук, М.В. Онай // XII наукова

конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг (ПМК-2019) : збірник тез доповідей. – К. : ФПМ КПІ ім. Ігоря Сікорського, 2019. – с. 85-91.

12. Witten, Ian H.; Frank, Eibe; Hall, Mark A. (30 January 2011). Data Mining: Practical Machine Learning Tools and Techniques (3 ed.). Elsevier
13. Indurkhya, N., and Damerau, F. (2010). Handbook Of Natural Language Processing, 2nd Edition. Boca Raton, FL: CRC Press
14. Sekine S. Named entities: recognition, classification and use [Text] / S. Sekine, E. Ranchhod // John Benjamins Publishing – 2009.
15. Nadeau D. A survey of named entity recognition and classification [Text] / D. Nadeau, S. Sekine // National Research Council Canada – 2006.
16. Collins M. Unsupervised models for named entity classification [Text] / M. Collins, Y. Singer // AT&T Labs-Research – 1999.
17. Li J. A Survey on Deep Learning for Named Entity Recognition [Text] / Jing Li et al. // Available: https://arxiv.org/pdf/1812.09449.pdf – 2017.
18. Coursera. Рекурентні нейронні мережі [Електронний ресурс]. — Дата візиту 13.06.2019. — Режим доступу до ресурсу: https://www.coursera.org/learn/nlp-sequence-models/lecture/ftkzt/recurrent-neural-network-model.
19. Система автоматичного розгортання проектів "Docker" [Електронний ресурс]. — Дата візиту 21.08.2019. — Режим доступу до ресурсу: https://docs.docker.com/.
20. Named Entity Recognition with SpaCy [Електронний ресурс]. — Режим доступу: https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da. — Дата доступу : 13.09.2019
21. Голосовий асистент «Wit AI» [Електронний ресурс]. — Режим доступу: https://wit.ai. — Дата доступу : 15.09.2019
22. Голосовий асистент «Amazon Alexa» [Електронний ресурс]. — Режим доступу: https://developer.amazon.com/en-US/docs/alexa/alexa-voice-service/api-overview.html. — Дата доступу : 15.09.2019

23. Голосовий асистент «Siri» [Електронний ресурс]. — Режим доступу: https://developer.apple.com/siri/. — Дата доступу : 15.09.2019
24. Голосовий асистент «Cortrana» [Електронний ресурс]. — Режим доступу: https://docs.microsoft.com/en-us/cortana/skills/. — Дата доступу : 15.09.2019
25. Голосовий асистент «MyCroft» [Електронний ресурс]. — Режим доступу: https://mycroft.ai/. — Дата доступу : 18.09.2019

ДОДАТКИ

Додаток 1
Лістинг програми

Лістинг 1. spacy_ner_initializer.py

```
from __future__ import unicode_literals, print_function
import pickle
import plac
import random
from pathlib import Path
import spacy
from spacy.util import minibatch, compounding


LABEL = ['variable_name', 'return_type', 'param_type', 'function_name',
'command', 'class_name', 'filename']

with open (dataset, 'rb') as fp:
    TRAIN_DATA = pickle.load(fp)

@plac.annotations(
    model=("Model name. Defaults to blank 'en' model.", "option", "m",
str),
    new_model_name=("New model name for model meta.", "option", "nm",
str),
    output_dir=("Optional output directory", "option", "o", Path),
    n_iter=("Number of training iterations", "option", "n", int))

def train(model=None, new_model_name='new_model', output_dir=None,
n_iter=10):
    if model is not None:
        nlp = spacy.load(model)
        print("Loaded model '%s'" % model)
    else:
        nlp = spacy.blank('en')
        print("Created blank 'en' model")
    if 'ner' not in nlp.pipe_names:
        ner = nlp.create_pipe('ner')
        nlp.add_pipe(ner)
    else:
        ner = nlp.get_pipe('ner')

    for i in LABEL:
        ner.add_label(i)

    if model is None:
        optimizer = nlp.begin_training()
    else:
        optimizer = nlp.entity.create_optimizer()

    other_pipes = [pipe for pipe in nlp.pipe_names if pipe != 'ner']
    with nlp.disable_pipes(*other_pipes):
        for itn in range(n_iter):
            random.shuffle(TRAIN_DATA)
            losses = {}
            batches = minibatch(TRAIN_DATA, size=compounding(4., 32.,
1.001))
            for batch in batches:
                texts, annotations = zip(*batch)
                nlp.update(texts, annotations, sgd=optimizer, drop=0.35,
                           losses=losses)
            print('Losses', losses)

    doc = nlp(test_text)
    print("Entities in '%s'" % test_text)
    for ent in doc.ents:
```

```
        print(ent.label_, ent.text)

    if output_dir is not None:
        output_dir = Path(output_dir)
        if not output_dir.exists():
            output_dir.mkdir()
        nlp.meta['name'] = new_model_name
        nlp.to_disk(output_dir)
        print("Saved model to", output_dir)
```

Лістинг 2. text_to_speech_processor.py

```
import pyaudio
import requests

class TextToSpeechProcessor:
    RATE = 22050
    SAMPWIDTH = 2
    NCHANNELS = 1
    ACCEPT = 'audio/wav'

    def __init__(self, user, password, voice = 'english',
                 url = 'local',
                 chunk = 2048):
        self.user = user
        self.password = password
        self.voice = voice
        self.url = url
        self.chunk = int(chunk)

    def play(self, text):
        req = requests.get(self.url + "/v1/synthesize",
                           auth=(self.user, self.password),
                           params={'text': text, 'voice': self.voice,
'accept': self.ACCEPT},
                           stream=True, verify=False)

        p = pyaudio.PyAudio()

        stream = p.open(format=p.get_format_from_width(self.SAMPWIDTH),
                        channels=self.NCHANNELS,
                        rate=self.RATE,
                        output=True)
        bytesRead = 0
        dataToRead = b''
        for data in req.iter_content(1):
            dataToRead += data
            bytesRead += 1
            if bytesRead % self.chunk == 0:
                stream.write(dataToRead)
                dataToRead = b''
        stream.stop_stream()
        stream.close()
        p.terminate()
        print("Synthesize complete")
```

Лістинг 3. spacy_recognizer.py

```
from __future__ import unicode_literals, print_function

from data import getData

import ujson as json
```

```
import pathlib
import random

import spacy
from spacy.pipeline import EntityRecognizer
from spacy.gold import GoldParse
from spacy.tagger import Tagger
from spacy.vocab import Vocab



nlp = spacy.load('en')

def train_ner(nlp, train_data, entity_types):
    for raw_text, _ in train_data:
        doc = nlp.make_doc(raw_text.decode('utf-8'))
        for word in doc:
            _ = nlp.vocab[word.orth]
    ner = EntityRecognizer(nlp.vocab, entity_types=entity_types)
    for itn in range(5):
        random.shuffle(train_data)
        for raw_text, entity_offsets in train_data:
            doc = nlp.make_doc(raw_text.decode('utf-8'))
            gold = GoldParse(doc, entities=entity_offsets)
            ner.update(doc, gold)
    ner.model.end_training()
    return ner


def main(model_dir=None):
    if model_dir is not None:
        model_dir = pathlib.Path(model_dir)
        if not model_dir.exists():
            model_dir.mkdir()
        assert model_dir.is_dir()

    nlp = spacy.load('en', parser=False, entity=False, add_vectors=False)

    if nlp.tagger is None:
        print('---- WARNING ----')
        print('Data directory not found')
         print('Using feature templates for tagging')
        print('-----------------')
        nlp.tagger = Tagger(nlp.vocab, features=Tagger.feature_templates)

    train_data =getData()
    ner = train_ner(nlp, train_data, topics)

    doc = nlp.make_doc(data)
    nlp.tagger(doc)
    ner(doc)
    for word in doc:
        print(word.text, word.orth, word.lower, word.tag_,
word.ent_type_, word.ent_iob)

    if model_dir is not None:
        with (model_dir / 'config.json').open('wb') as file_:
            json.dump(ner.cfg, file_)
        ner.model.dump(str(model_dir / 'model'))
        if not (model_dir / 'vocab').exists():
            (model_dir / 'vocab').mkdir()
        ner.vocab.dump(str(model_dir / 'vocab' / 'lexemes.bin'))
        with (model_dir / 'vocab' /
'strings.json').open('w',encoding='utf8') as file_:
```

```
            ner.vocab.strings.dump(file_)

def predict(query):
    nlp = spacy.load('en', parser=False, entity=False, add_vectors=False)
    vocab_dir = pathlib.Path('ner/vocab')
    with (vocab_dir / 'strings.json').open('r',encoding='utf8') as file_:
        nlp.vocab.strings.load(file_)
    nlp.vocab.load_lexemes(vocab_dir / 'lexemes.bin')


    ner = EntityRecognizer.load(pathlib.Path("ner"), nlp.vocab,
require=False)
    doc = nlp.make_doc(query)
    #nlp.tagger(doc)
    ner(doc)
    for word in doc:
        print(word.text, word.orth, word.lower, word.ent_type_)

    for word in doc:
        if word.ent_type_:
            print ('word -> {} and entity->
{}'.format(word.text,word.ent_type_))
        print(word.text, word.orth, word.lower, word.ent_type_)
```

Лістинг 4. ast_node_visitor.py

```
class AnalysisNodeVisitor(ast.NodeVisitor):

    def __init__(self,rootNode = None):
        self._modules = []
        self._classes = []
        self._functions = []
        self._variables = []
        self._imports = []
        self._rootNode = rootNode
        self._parentNode = rootNode
        self._level = 0

    @property
    def rootNode(self):
        return self._rootNode

    @property
    def imports(self):
        return self._imports

    @property
    def functions(self):
        return self._functions

    @property
    def variables(self):
        return self._variables

    @property
    def classes(self):
        return self._classes

    def visit_Import(self,node):
        for name in node.names:
```

```
            importNode = Node(attributes =
{'type':'import','names':map(lambda x:x.name,node.names)},parent =
self._parentNode)
            self._imports.append(importNode)
        ast.NodeVisitor.generic_visit(self, node)

    def visit_ImportFrom(self,node):
        for name in node.names:
            importNode = Node(attributes =
{'line_number':node.lineno,'type':'from_import','module':node.module,'nam
es':map(lambda x:x.name,node.names)},parent = self._parentNode)
            self._imports.append(importNode)
        ast.NodeVisitor.generic_visit(self, node)

    def visit_Assign(self,node):
        for target in node.targets:
            self._add_target_to_variables(target)
        ast.NodeVisitor.generic_visit(self, node)

    def visit_AssignAug(self,node):
        self._add_target_to_variables(node.target)
        ast.NodeVisitor.generic_visit(self, node)

    def _add_target_to_variables(self,target):
        if hasattr(target,'value'):
            self._add_target_to_variables(target.value)
        elif hasattr(target,'id'):
            if not target.id in self._variables and not target.id ==
"self":
                variableNode = Node(attributes =
{'type':'variable','name':target.id},parent = self._parentNode)
                self._variables.append(variableNode)

    def visit_FunctionDef(self,node):
        body = node.body
        functionNode = Node(attributes =
{'type':'function','name':node.name,'start_line':body[0].lineno,'end_line
':_get_last_line_number(body),'docstring':ast.get_docstring(node)},parent
= self._parentNode)
        self._functions.append(functionNode)
        oldParent = self._parentNode
        self._parentNode = functionNode
        ast.NodeVisitor.generic_visit(self, node)
        self._parentNode = oldParent

    def visit_ClassDef(self,node):
        body = node.body
        classNode = Node(attributes =
{'type':'class','name':node.name,'start_line':body[0].lineno,'end_line':_
get_last_line_number(body),'docstring':ast.get_docstring(node)},parent =
self._parentNode)
        self._classes.append(classNode)
        oldParent = self._parentNode
        self._parentNode = classNode
        ast.NodeVisitor.generic_visit(self, node)
        self._parentNode = oldParent
```

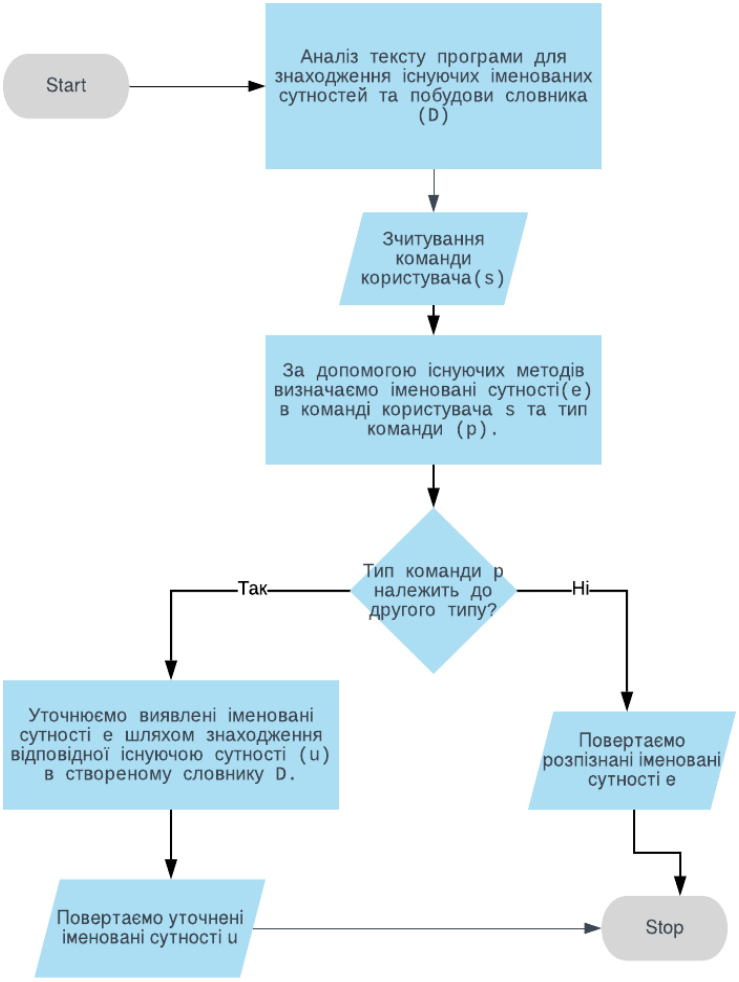
Лістинг 5. ElasticSearchLogger.java

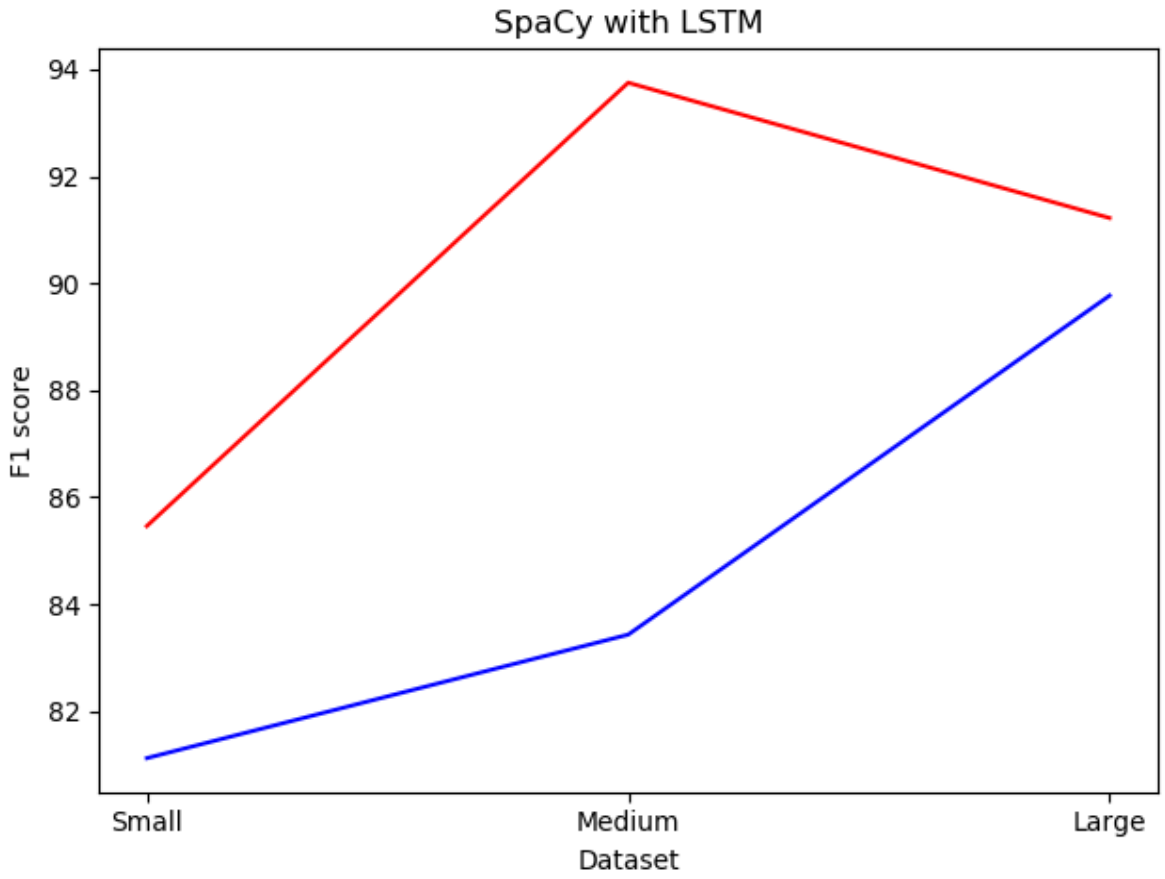
```
import com.fasterxml.jackson.databind.ObjectMapper;
import org.elasticsearch.ElasticsearchException;
import org.elasticsearch.action.get.GetRequest;
import org.elasticsearch.action.get.GetResponse;
```

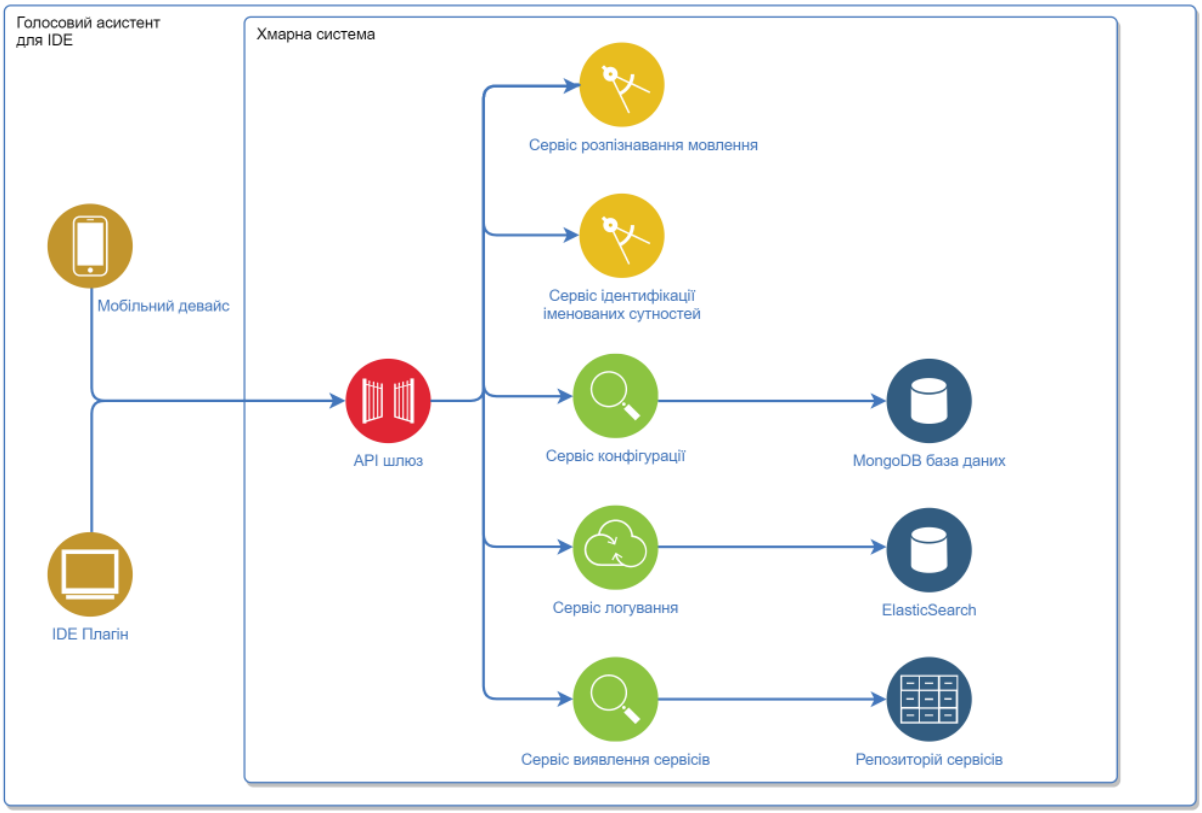
```
import org.elasticsearch.action.index.IndexRequest;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.common.xcontent.XContentType;
import org.elasticsearch.rest.RestStatus;

import java.io.IOException;
import java.util.UUID;

public class GetApi {

    private final RestHighLevelClient client;
    private final String index;
    private final String type;
    private final ObjectMapper mapper;

    public GetApi(final RestHighLevelClient client, final String index,
final String type) {
        this.client = client;
        this.index = index;
        this.type = type;
        this.mapper = new ObjectMapper();
    }

    public void getRequest() throws IOException {
        final User user = prepareUser();
        createUserRequest(user);

        final GetRequest request = new GetRequest(index, type,
user.getId());
        final GetResponse response = client.get(request,
Application.prepareAuthHeader());
        final User responseUser =
mapper.readValue(response.getSourceAsString(), User.class);
        assert user.equals(responseUser);
    }

    public void getRequestInvalidIndex() throws IOException {
        final GetRequest request = new GetRequest(index_name, type,
UUID.randomUUID().toString());
        try {
            client.get(request, Application.prepareAuthHeader());
        } catch (ElasticsearchException e) {
            if (e.status() == RestStatus.NOT_FOUND) {
                System.out.println(e.getDetailedMessage());
            }
        }
    }

    private void createUserRequest(final User user) throws IOException {
        final IndexRequest indexRequest = new IndexRequest(index, type,
user.getId());
        indexRequest.source(mapper.writeValueAsString(user),
XContentType.JSON);

        client.index(indexRequest, Application.prepareAuthHeader());
    }

    private User prepareUser() {
        return User.getBuilder()
                .setId(UUID.randomUUID().toString())
                .setFirstname(MDC.getName())
                .setLastname(MDC.getSurname())
                .build();
    }}
```

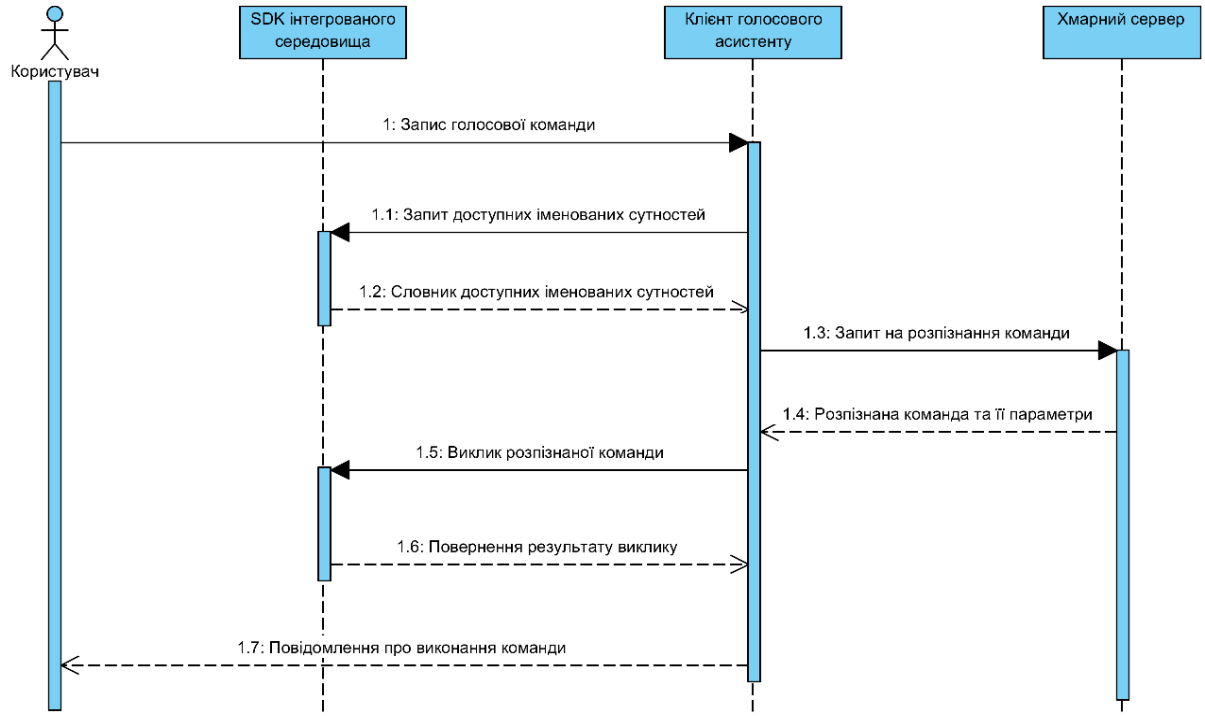
Додаток 2
Копії графічних матеріалів


Всі наявні середовища розробки не мають рішень для цього сегменту користувачів
Відсутність засобів для написання коду для людей з обмеженими можливостями
Низька швидкість введення інформації з клавіатури
Концентрування програміста на деталях реалізації, а не на вирішенні проблеми
Неефективні засоби введення інформації
Неефективність інтегрованих середовищ розробки програмного забезпечення
Неможливість написання коду на мобільних девайсах
Низька швидкість введення інформації
Немає наявних інтегрованих середовищ розробки


Start
Аналіз тексту програми для знаходження існуючих іменованих сутностей та побудови словника (D)
Зчитування команди користувача(s)
За допомогою існуючих методів визначаємо іменовані сутності(e) в команді користувача s та тип команди (p).
Тип команди p належить до другого типу?
Так
Ні
Уточнюємо виявлені іменовані сутності e шляхом знаходження відповідної існуючою сутності (u) в створеному словнику D.
Повертаємо розпізнані іменовані сутності e
Повертаємо уточнені іменовані сутності u
Stop


SpaCy with LSTM
F1 score
94
92
90
88
86
84
82
Small
Medium
Large
Dataset


Голосовий асистент
для IDE
Хмарна система
Сервіс розпізнавання мовлення
Мобільний девайс
Сервіс ідентифікації
іменованих сутностей
API шлюз
Сервіс конфігурації
MongoDB база даних
IDE Плагін
Сервіс логування
ElasticSearch
Сервіс виявлення сервісів
Репозиторій сервісів


Користувач
SDK інтегрованого середовища
Клієнт голосового асистенту
Хмарний сервер
1: Запис голосової команди
1.1: Запит доступних іменованих сутностей
1.2: Словник доступних іменованих сутностей
1.3: Запит на розпізнання команди
1.4: Розпізнана команда та її параметри
1.5: Виклик розпізнаної команди
1.6: Повернення результату виклику
1.7: Повідомлення про виконання команди


Main
Main.java
Voice control
package com.company;
public class Main {
public static void main(String[] args) {
String a;
a = "five";
System.out.println(a);
}
}

Додаток 3
Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

ГОЛОСОВИЙ АСИСТЕНТ ДЛЯ ІНТЕГРОВАНИХ СЕРЕДОВИЩ РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Виконав: Сахарчук Тарас Юрійович

Науковий керівник: к.т.н., доцент Онай Микола Володимирович

Консультант: к.т.н., доцент Сулема Євгенія Станіславівна

Київ – 2019

АКТУАЛЬНІСТЬ ТЕМИ

Засоби, які інженери використовують при розробленні програмного забезпечення відіграють велику роль і впливають на якість тексту програми та час необхідний для його створення.

Об’єкт дослідження: процес використання мовлення для розроблення програмного забезпечення.

Предмет дослідження: методи та підходи до вирішення проблем розпізнавання людського мовлення, ідентифікації іменованих сутностей в неструктурованому тексті та генерації тексту програми.

Наукове завдання: удосконалити існуючі підходи до ідентифікації іменованих сутностей тексту програми та з їх використанням побудувати голосовий асистент для інтегрованих середовищ розроблення.

Мета дослідження: покращення точності існуючих методів та підходів до ідентифікації іменованих сутностей тексту програми.

Окремі завдання

1. Здійснити аналіз існуючих рішень.
2. Оптимізувати існуючі методи та підходи до ідентифікації іменованих сутностей тексту програми шляхом врахування особливостей обраної предметної області.
3. Розробити масштабовану, модульну архітектуру системи.
4. Розробити та протестувати програмне забезпечення для голосового асистенту.
5. Описати бізнес-модель, що дозволить представити на ринку повноцінний програмний продукт.

АНАЛІЗ ІСНУЮЧИХ ПРОБЛЕМ

АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ ЗАСОБІВ

	Alexa	Google Assistant	Cortrana	VoiceCode	Silvius	Vocola
Орієнтованість на написання тексту програми голосом	–	–	–	+	+	+
Плаформність або модульність системи	+	+	+	–	–	–
Якість документації	+	+	+	+	–	–
Можливість надання користувачем високорівневих команд	+	+	+	–	–	–
Не потребує витрат	–	–	–	–	+	+

ЗАДАЧА РОЗРОБЛЕННЯ ГОЛОСОВОГО АСИСТЕНТУ

- Розпізнавання мовлення. Перетворення мовлення користувача в текст (STT).
- Використання методів обробки природного мовлення користувачів та підходів в області глибокого машинного навчання для виявлення в тексті команд, заданих користувачем, та їх параметрів (ідентифікація іменованих сутностей).
- На основі заданої команди та її параметрів генерувати відповідний текст програми для вибраної мови програмування або виконувати відповідну функцію в інтегрованому середовищі розроблення програмного забезпечення.

ІСНУЮЧІ ПІДХОДИ ДО РОЗПІЗНАВАННЯ МОВЛЕННЯ ТА ІДЕНТИФІКАЦІЇ ІМЕНОВАНИХ СУТНОСТЕЙ

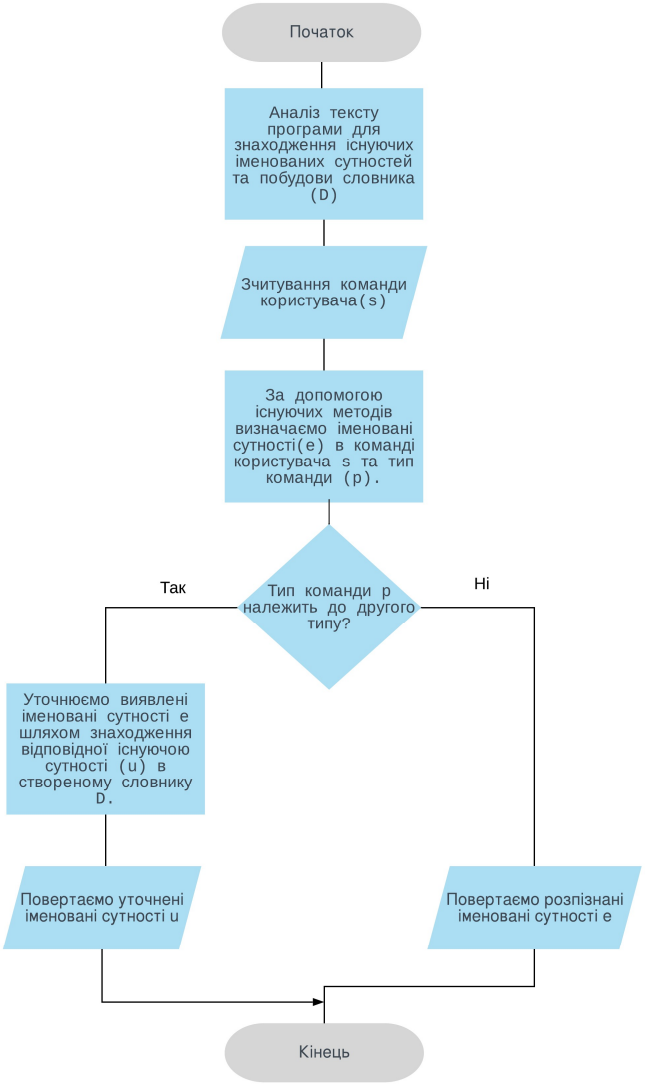
- Навчання з вчителем (HMM, CRF)
- Навчання без вчителя (K-means)
- Напівавтоматичне навчання
- Глибоке машинне навчання (CNN, RNN)

ПРИКЛАД РОЗПІЗНАВАННЯ ІМЕНОВАНИХ СУТНОСТЕЙ

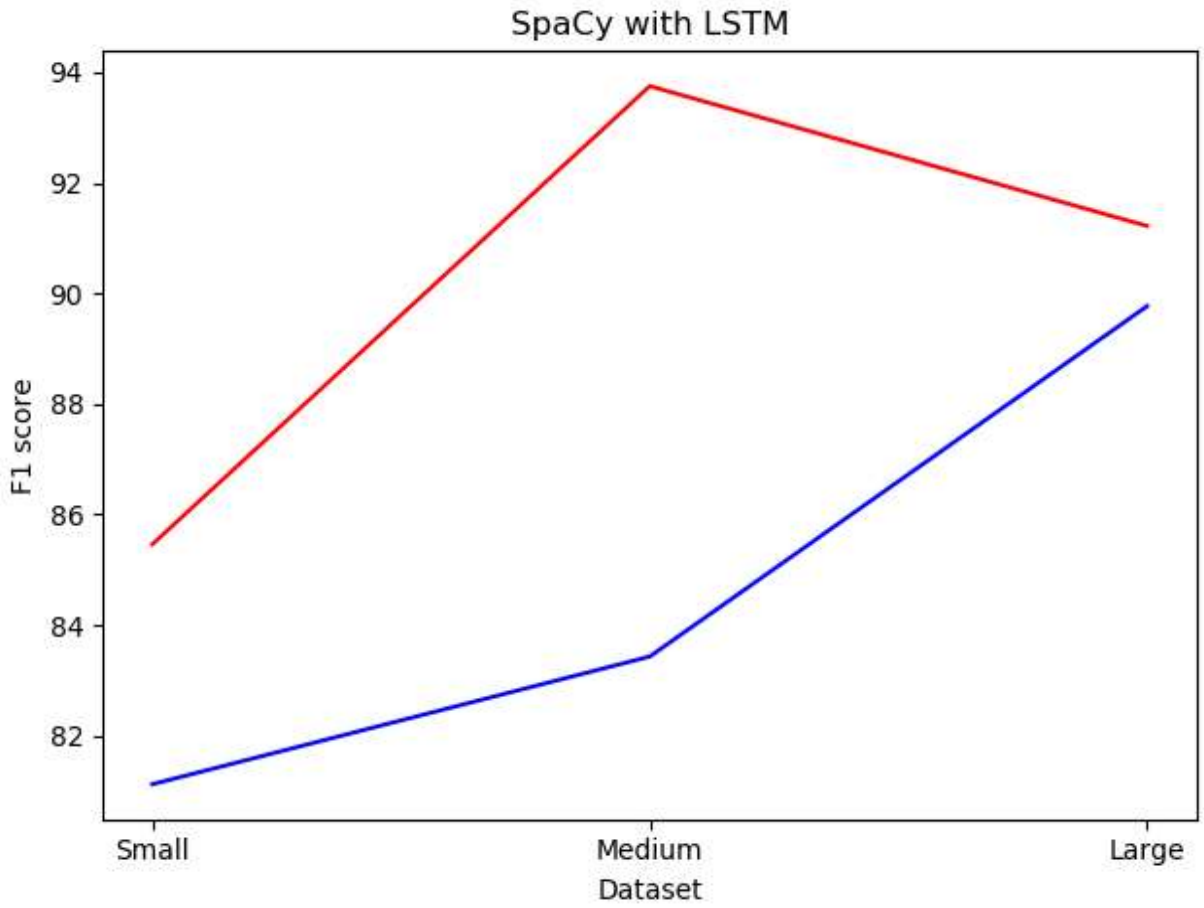
МОДИФІКОВАНИЙ МЕТОД РОЗПІЗНАВАННЯ ІМЕНОВАНИХ СУТНОСТЕЙ ТЕКСТУ ПРОГРАМИ

Для покращення точності розпізнавання іменованих сутностей в командах користувача доцільно використовувати інформацію, яка наявна в контексті тексту програми, а саме доступні назви змінних, операторів, функцій, модулів, файлів тощо.

СХЕМА МОДИФІКОВАНОГО МЕТОДУ РОЗПІЗНАВАННЯ СУТНОСТЕЙ



ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ МОДИФІКОВАНОГО МЕТОДУ



АРХІТЕКТУРА СТВОРЕНОЇ СИСТЕМИ ГОЛОСОВОГО АСИСТЕНТУ ДЛЯ ІНТЕГРОВАНИХ СЕРЕДОВИЩ

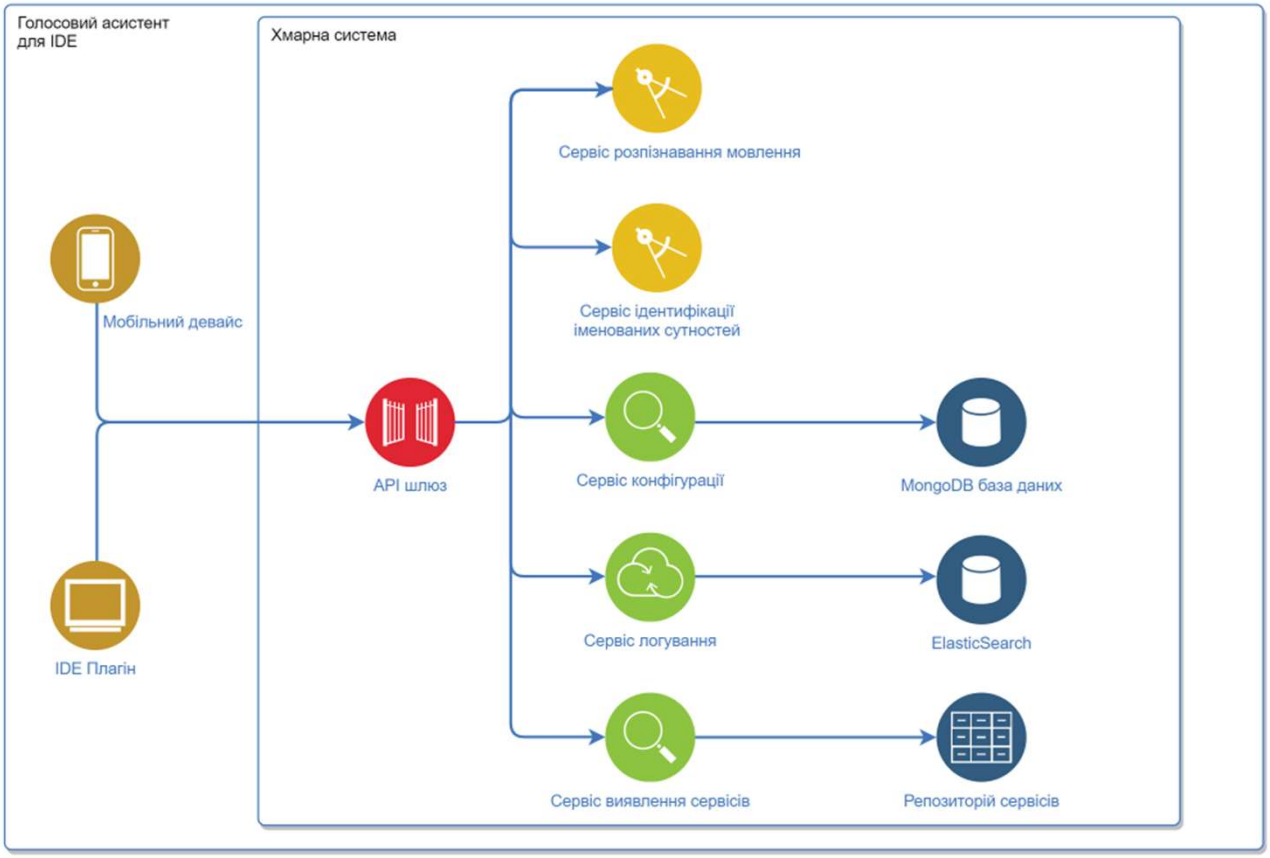
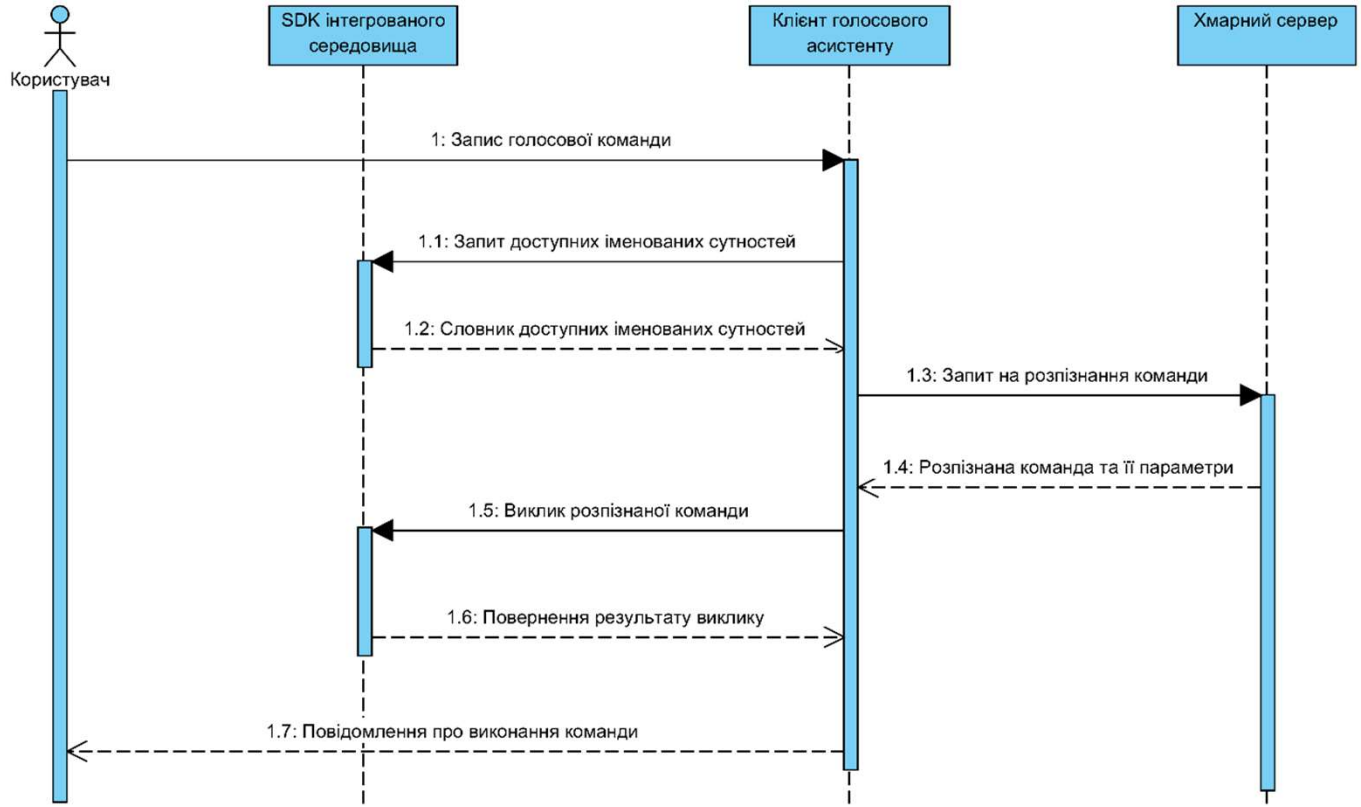


СХЕМА ОБРОБКИ ГОЛОСОВОЇ КОМАНДИ КОРИСТУВАЧА



АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

- Система складається з клієнта, API шлюзу та 7 сервісів
- Клієнт представлений у вигляді плагіну для IntelliJ IDEA
- Система розпізнає команди користувача англійською мовою та надає можливість створення та редагування змінних, функцій, класів, файлів, а також керуванням середовищем розроблення
- Покращена точність розпізнавання команд користувача завдяки використанню модифікованого методу ідентифікації іменованих сутностей.
- Система покрита юніт-тестами, а також було здійснено ручне тестування функціональності.

РЕКОМЕНДАЦІЇ ЩОДО ПОДАЛЬШОГО ВДОСКОНАЛЕННЯ

- Розширення списку підтримуваних інтегрованих середовищ розроблення програмного забезпечення, мов програмування та команд користувача
- Реалізація клієнту на базі мобільних платформ
- Розширення клієнту для автономної роботи у випадку втрати з’єднання з сервером

ІНТЕРФЕЙС ПЛАГІНУ ДЛЯ ГОЛОСОВОГО АСИСТЕНТА

БІЗНЕС-МОДЕЛЬ

<table>
<tr><th>Ключові партнери</th><th>Ключова діяльність</th><th>Цінністна пропозиція</th><th>Відношення з клієнтами</th><th>Сегменти користувачів</th></tr>
<tr><td rowspan="2">1. Постачальники серверного та комп`ютерного обладнання.
2. Компанії, які використовують веб-застосунки.
3. Інвестори.</td><td>Розроблення голосового асистенту для інтегрованих середовищ.</td><td rowspan="2">Розроблена система для голосового управління інтегрованим середовищем розроблення.</td><td>Побудова лояльності</td><td rowspan="2">1. Компанії, які займаються розробкою.
2. Розробники програмного забезпечення</td></tr>
<tr><td>Ключові ресурси
1. Персонал.
2. Серверне обладнання.</td><td>Канали
1. Соціальні мережі.
2. Контекстна реклама.</td></tr>
<tr><th colspan="2">Структура витрат</th><th colspan="3">Джерела доходу</th></tr>
<tr><td colspan="2">Фіксовані витрати (оренда приміщення, з/п персоналу)</td><td colspan="3">Дохід від продажу ліцензій</td></tr>
</table>

НАУКОВО-ІНОВАЦІЙНА НОВИЗНА

- Удосконалено існуючі підходи до розпізнавання іменованих сутностей тексту програми шляхом врахування контекстної інформації для обраної предметної області.
- Запропоновано модифікований метод розпізнавання іменованих сутностей тексту програми, який полягає в уточненні результатів отриманих стандартними методами шляхом використання інформації про вже введені сутності в тексті програми та забезпечує підвищення точності в середньому на 5% F1 оцінки.

ВИСНОВКИ

1. Проаналізовано існуючі програмні рішення.
2. Запропоновано модифікований метод для вирішення поставленої задачі, який враховує особливості обраної доменної області та дозволяє підвищити точність розпізнавання іменованих сутностей на 5% (F1 score).
3. Розроблено масштабовану, модульну архітектуру системи.
4. Реалізовано та покрито тестами програмне забезпечення для голосового асистента.
5. Проведено аналіз ринку та визначено подальші перспективи проекту, побудовано бізнес-модель.

АПРОБУВАННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

1. XII наукова конференція магістрантів та аспірантів «Прикладна математика та комп’ютинг» (ПМК-2019).

ПЕРЕВІРКА НА УНІКАЛЬНІСТЬ

Унікальність текстової частини магістерської дисертації становить 99.35% при перевірці системою «Unicheck».

Дякую за увагу!